

(2)

NAVAL POSTGRADUATE SCHOOL

Monterey, California

DMC EFILE COPY



THESIS

DTIC
ELECTED
04 JAN 1989
S E D

A SYSTEM COSTS PLANNING DECISION SUPPORT SYSTEM

by

Craig L. Riddle

September 1988

Thesis Advisor:

Gerald L. Pauder

Approved for public release; distribution is unlimited.

89 1 04 022

AD-A202 068

Unclassified

Security Classification of this page

REPORT DOCUMENTATION PAGE

1a Report Security Classification Unclassified		1b Restrictive Markings	
2a Security Classification Authority		3 Distribution Availability of Report Approved for public release; distribution is unlimited.	
2b Declassification/Downgrading Schedule		5 Monitoring Organization Report Number(s)	
4 Performing Organization Report Number(s)		7a Name of Monitoring Organization Naval Postgraduate School	
6a Name of Performing Organization Naval Postgraduate School	6b Office Symbol (If Applicable) 37	7b Address (city, state, and ZIP code) Monterey, CA 93943-5000	
8a Name of Funding/Sponsoring Organization	8b Office Symbol (If Applicable)	9 Procurement Instrument Identification Number	
8c Address (city, state, and ZIP code)		10 Source of Funding Numbers Program Element Number Project No Task No Work Unit Accession No	

11 Title (Include Security Classification) A System Costs Planning Decision Support System

12 Personal Author(s) Craig L. Riddle

13a Type of Report Master's Thesis	13b Time Covered From To	14 Date of Report (year, month, day) 1988 September	15 Page Count 96
---------------------------------------	-----------------------------	--	---------------------

16 Supplementary Notation The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

17 Cosati Codes			18 Subject Terms (continue on reverse if necessary and identify by block number) Decision Support System, Multi-Attribute Utility theory, Prototyping
Field	Group	Subgroup	

19 Abstract (continue on reverse if necessary and identify by block number)

This study undertakes the task of analysis and design to create a prototype microcomputer-based decision support system for cost planning. System acquisition cost planning is a complex process in which a variety of ill-defined, often conflicting variables influence the decision to be made. Multi-Attribute Utility Theory (MAUT) offers a Multicriterion decision method to incorporate and quantify these variables in the search for an optimal solution. A decision is defined by the options among which one must choose, the possible outcomes, or consequences. Typically, there exists a measurable preference among various choices when making a decision. This preference is called "utility." Microeconomic marginal analysis applied to utility curves generated from MAUT data derivation reveals insights to decision risk assessment and cost planning limitations. In a decision with preferences spread among several goals, the utilities may be assigned different weights to determine overall utility value. This theory of weighted utility is the basis for this prototype. This study envelopes user-oriented analysis and design of a prototype. The microcomputer code is developed for in-house use by decision makers, thus facilitating the management decision process more cost effectively and in less time.

20 Distribution/Availability of Abstract <input checked="" type="checkbox"/> unclassified/unlimited <input type="checkbox"/> same as report <input type="checkbox"/> DTIC users	21 Abstract Security Classification Unclassified
22a Name of Responsible Individual Professor Gerald L. Paurer	22b Telephone (Include Area code) (408) 646-2938
	22c Office Symbol 55Pa

Approved for public release; distribution is unlimited.

A System Costs Planning Decision Support System

by

Craig L. Riddle
Lieutenant, United States Navy
B.S., United States Naval Academy, 1981

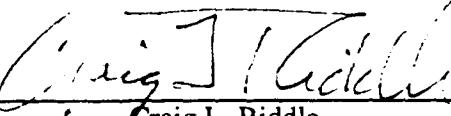
Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN INFORMATION SYSTEMS

from the

NAVAL POSTGRADUATE SCHOOL
September 1988

Author:

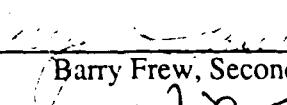


Craig L. Riddle

Approved by:



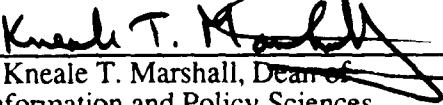
Gerald L. Pauler, Thesis Advisor



Barry Frew, Second Reader



David R. Whipple, Chairman,
Department of Administrative Sciences



Kneale T. Marshall, Dean of
Information and Policy Sciences

ABSTRACT

This study undertakes the task of analysis and design to create a prototype microcomputer-based decision support system for cost planning. System acquisition cost planning is a complex process in which a variety of ill-defined, often conflicting variables influence the decision to be made. Multi-Attribute Utility Theory (MAUT) offers a Multicriterion decision method to incorporate and quantify these variables in the search for an optimal solution. A decision is defined by the options among which one must choose, the possible outcomes, or consequences. Typically, there exists a measurable preference among various choices when making a decision. This preference is called "utility." Microeconomic marginal analysis applied to utility curves generated from MAUT data derivation reveals insights to decision risk assessment and cost planning limitations. In a decision with preferences spread among several goals, the utilities may be assigned different weights to determine overall utility value. This theory of weighted utility is the basis for this prototype. This study envelopes user-oriented analysis and design of a prototype. The microcomputer code is developed for in-house use by decision makers, thus facilitating the management decision process more cost effectively and in less time.

Accession For	
NTIS GRA&I	<input checked="" type="checkbox"/>
DTIC TAB	<input type="checkbox"/>
Unpublished	<input type="checkbox"/>
Justification	
By _____	
Distribution/	
Availability Codes	
Avail and/or	
Dist	Special
A-1	



TABLE OF CONTENTS

I. INTRODUCTION	1
A. BACKGROUND.....	1
B. PURPOSE	1
C. SCOPE	2
D. METHODOLOGY	2
II. SURVEY OF DSS LITERATURE.....	4
A. DSS FRAMEWORK AND USAGE.....	5
B. DECISION-CENTERED DECISION MAKING	6
III. A SYSTEM COSTS PLANNING-DSS	9
A. SUMMARY DESCRIPTION.....	9
B. PRELIMINARY CALCULATIONS	13
C. BUILDING THE UTILITY CURVE.....	14
D. COST RANGE CALCULATIONS AND RESULTS	17
IV. THE SCP-DSS SYSTEM DESIGN.....	19
A. USER-ORIENTED DESIGN.....	19
1. User-Oriented Analysis.....	20
2. User-Oriented Implementation.....	21
B. FUNCTIONS OF THE SCP-DSS.....	21

C. DATA FLOW OF THE SCP-DSS	21
D. THE DATA COMPONENT	22
E. DIALOG COMPONENT.....	24
1. User Interface.....	25
a. Standard Screen.....	25
b. Input/Output	25
c. Reports.....	26
d. Help	27
2. Component Linkage.....	27
F. SCP-DSS MODEL COMPONENT.....	27
1. Matrix Building Functions.....	27
a. Cost Matrix	27
b. Utility Matrix	28
c. Cumulative Costs Matrix.....	28
d. Weighted Utility Matrix	28
2. Model Execution.....	28
3. Dialog Interface	29
4. Data Interface.....	30
 V. PROTOTYPE IMPLEMENTATION OF THE SCP-DSS.....	31
A. PROTOTYPING LANGUAGE.....	33
B. PROTOTYPING PROBLEMS.....	33
1. User Requirements	33
2. Programming Environment.....	34

VI. CONCLUSIONS AND RECOMMENDATIONS	36
A. CONCLUSIONS.....	36
B. RECOMMENDATIONS FOR FURTHER STUDY.....	37
APPENDIX A	MULTI-ATTRIBUTE UTILITY THEORY
APPENDIX B	REFERENCES ON MICROECONOMIC MARGINAL THEORY.....
APPENDIX C	DATA FLOW DIAGRAMS
APPENDIX D	FIRST PROGRAM LISTING IN PASCAL
APPENDIX E	SECOND PROGRAM LISTING IN PASCAL.....
LIST OF REFERENCES.....	84
INITIAL DISTRIBUTION LIST.....	87

ACKNOWLEDGMENTS

At the completion of this research the author wishes to express his gratitude and sincere appreciation to G. Pauler and B. Frew for their assistance, support, and guidance.

Furthermore, he would like to convey his sincere appreciation to M. Mayer for his recursive perspective toward the coding of this project.

Finally, the author dedicates this thesis to his loved ones and family for their continued support and encouragement that have helped him in his efforts for education, self improvement, and happiness.

I. INTRODUCTION

A. BACKGROUND

Decision support systems (DSSs) began in the 1960s as a concept for integration of computer processing with modeling tools such as linear programming and/or simulation as an interactive tool [Ref. 1]. The idea of a human-machine interactive system in which the decision task was divided between the human and the machine in a synergistic problem-solving system was emphasized. The intent was that each component did those parts of the task for which it was best suited [Ref. 2]. A DSS can be represented as an interactive computer-based system that helps decision makers utilize data and models to solve unstructured problems [Ref. 3].

One such environment for DSS is that of System Costs Planning (SCP). A DSS can provide computational and analytical support in the SCP process to complement judgment, expertise, experience, and insight. As such, it uses Multi-Attribute Utility Theory (MAUT). MAUT is a method used to incorporate and quantify data variables to obtain a recommended decision. Turban and Meredith state briefly that "...there is a measurable preference among various choices available in risk situations." [Ref. 4] This "...measurable preference..." is called Utility. In a decision involving Multi-Attribute, and thus Multi-Criteria, utility values can be assessed for how well each attribute meets its associated criterion.

B. PURPOSE

The purpose of this thesis is to provide a comprehensive and responsive microcomputer Decision Support System to support SCP planners.

SCP-DSS users must understand MAUT as background to support their decision making (Appendix A contains a brief overview of MAUT). System use is correlated to its ability to support users' needs.

C. SCOPE

This thesis uses a percentage weighted criteria method of MAUT, for development of an integrated and graphic DSS for SCP tasks. An aircraft upgrade decision is used as an example application. Aircraft components specified for upgrade constitute the decision criteria. These criteria are also referred to as attributes (of the upgrade system), features, or decision variables. Each decision variable has multiple options from which a combination upgrade, consisting of one option per variable giving the highest utility score, is chosen. The cost parameter is the user-defined target for the SCP-DSS. It searches all possible combinations of options to isolate the combination with the highest utility score for the user's input target cost. The size of the user input matrices determines the complexity of the decision under consideration.

D. METHODOLOGY

The decision environment and an existing system were reviewed to ascertain how current methodologies and procedures worked in conjunction to aid decision makers. Analysis of the process identified a lengthy time element for problem solution plus a need for a easy-to-use local system. The decision makers identified relevant methodologies to aid the formulation of automation techniques. From this analysis, with the described problem areas and solution methodologies identified, the decision to build a prototype system was made. This allowed decision makers to provide constant input to the system design.

Using a prototype strategy consisting of a common language (PASCAL) base allowed the design of a low-cost system with immediate pay-off. A clear and concise

communication between the system users and the analyst kept the system design in a logical context until all areas of decision support were identified. Prototype development allowed the users to appreciate an almost immediately available automated solution for costs planning needs on the microcomputer.

II. SURVEY OF DSS LITERATURE

Many decision-making activities within an organization occur in an unstructured environment. This environment is characterized by constantly shifting goals, restructuring of priorities, and varying decision-making styles. Information is recognized as a major resource of any organization. Within a DSS environment, it supports the decision-making process. With managerial judgement being so critical to the decision process, a DSS must be designed to allow combining computerized output with managerial judgement. In essence, decision support systems are designed to support specific decision processes. Advances in technology, coupled with declining hardware costs, permit increasingly complex problem resolution on microcomputers. Design of a DSS requires a firm understanding of decision-making processes *within the organization*. An implicit assumption is that better data and more accurate models result in a better DSS. This idea, although relevant, does not guarantee that DSSs will function as important decision-making tools [Ref. 5]. Fundamentally, the main thrust of decision support systems is on problems for which there is sufficient structure for computer and mathematical (and statistical) models to be of value, but not to occlude the manager's essential judgment. DSS extends the range and capability of the decision process to help improve the decision maker's effectiveness. The relationship of the DSS to the manager is the creation of a supportive tool, under his control, which does not attempt to predefine objectives, automate the decision process, or impose solutions directly [Ref. 6].

DSS literature agrees upon the emergence of three main components: data, dialog, and model [Ref. 7]. A separate view of each component supports the development, design, implementation, and maintenance of the SCP-DSS. The data, dialog, and model components, with their varied complexities, based on user input, are not easily separated.

The model component, although not intricate in design, becomes complex when tied to a dialog component that is a complex and restrictive user interface. The model component is driven by the complexity of the data component through the dialog component.

A. DSS FRAMEWORK AND USAGE

To assess prototype system development, it is important to first structure a framework. As Sprague states, "A framework, in the absence of theory, is helpful in organizing a complex subject, identifying the relationships between the parts, and revealing the areas in which further developments will be required." [Ref. 7] Also, it is necessary that this framework be grounded in a firm realization of just "what exactly is a DSS." Sprague and Carlson define a DSS as a computer-based system that helps decision-makers confront ill-structured problems through direct interaction with data and analysis models [Ref. 3]. Using the MAUT model, with a microcomputer, experience, expertise, and insight as key elements, this definition guides analysis and design of this microcomputer-based DSS.

To establish a framework, a clear understanding of the DSS's information requirements concepts is key. Gorry and Scott-Morton define information requirements for Strategic Planning DSS types as having the following characteristics (by decision category) [Ref. 8]:

Characteristic of Information	Strategic Planning
Source	external
Scope	very wide
Level of Detail	aggregate
Time Horizon	future
Currency	quite old
Required Accuracy	quite low
Frequency of Use	infrequent

This Gorry and Scott-Morton table summarizes their notions of complex decision planning requirements. The example problem, described in Chapter III due to its complex information needs, alters this framework described in the following manner:

- Level of Detail that is *more detailed* versus aggregate;
- Currency that is relatively *near-term*; and
- Accuracy in *moderate to high* range.

This altered framework puts the requirements imposed here on the data, dialog, and model components into perspective to reveal how they have to be interrelated to increase their collective effectiveness. As decision makers come together to discuss formulation costs and utility values, their initial inputs to the matrices form the data component.

B. DECISION-CENTERED DECISION MAKING

Decision-making activities are at the center of the functions comprising the management process. The process of management is basically one of decision making. The DSS rationale for a combined mathematical/computerized approach to decision-making is the recognition by a decision maker that it is impossible, working alone, to evaluate all the factors for an effective decision. The microcomputer allows immediate and direct input or immediate retrievability of data to and from a database. Also, it allows the decision maker to solve mathematical and statistic style problems in a matter of minutes and hours instead of days, weeks, or months.

The Decision Centered approach to decision making is in contrast to the classical quantitative approach based on scientific method. The scientific method was originally formulated by Francis Bacon in the sixteenth century and elaborated by John Stuart Mill in the nineteenth century [9]. Its traditional steps are observation, definition of the problem, formulation of the hypothesis, experimentation, and verification. These steps are altered to adapt to the decision-making environment. This method is used to help decision makers

choose the best or “optimal” alternative, that is, one that balances the costs and benefits, along with some unknown factors. This is good practice in many cases, but many times the decision makers lack important information affecting a decision. With time constraints and decision anticipation affecting actions, many alternatives are overlooked in the decision process. These limitations restrict decision making, with the result being that of “satisficing.” The word “satisficing” means finding and selecting a satisfactory alternative (as opposed to the best one) that achieves a minimally acceptable solution [Ref. 10]. Decision makers should not select the first satisfactory alternative developed but rather should take the opportunity and time to develop other feasible alternatives.

Thierauf further defines the Decision-Centered method with the following:

An essential part of satisficing is the concept of bounded rationality. The fact that managers often make decisions without knowing all the alternatives available to them or possible consequences means that there is a limit to how logical or rational their decision can be. In organizational life, managers make the most logical decision they can, limited by their inadequate information and by their ability to utilize that information, thereby resulting in bounded rationality. Within bounded rationality, rather than make the best or ideal decision, managers more realistically settle for a decision that will satisfice rather than optimize. [Ref. 11]

This satisficing approach does not mean that decision makers cannot obtain the best possible solution. It is just that at some point it becomes too expensive in terms of time and money to gain the additional information needed. These are realities in any decision maker’s world, and Herbert Simon’s decision centered method [Ref. 12] is outlined as follows:

- Step 1: Intelligence—This is the data-gathering phase in which the decision maker seeks information to define the problem more clearly and provide some input to the solution process.
- Step 2: Design—The second step centers on inventing, developing, and analyzing possible courses of action. It involves manipulation of the data obtained to develop various alternative solutions to the problem.
- Step 3: Choice—This task is one of evaluating alternatives. This phase of the problem-solving process also requires selection of the best among the alternatives developed in the design phase.

Step 4: Implementation—This step puts the chosen solution into effect. In essence, the best alternative selected in the prior step is placed into operation.

Step 5: Control—The fifth step is monitoring the outcome and making necessary adjustments. This last step links back to the first step, intelligence, by recognizing any new problems that arise and need to be solved.

The foregoing steps allow the decision makers to explore all possibilities within a semistructured environment. To compare them on the same basis for an optimum answer may be too costly and time consuming using a quantitative method. The focus of decision support systems is on the semistructured and unstructured problem. The SCP-DSS user seeks the optimal decision with the best intelligence, design, and choice method available. Simon's Decision-Centered approach as applied to this thesis covers only the first three steps—those of intelligence, design, and choice. Implementation and control are follow-on stages past which the SCP-DSS is of minor assistance.

III. A SYSTEM COSTS PLANNING-DECISION SUPPORT SYSTEM (SCP-DSS) MODEL

A Costs Planning example to illustrate this SCP-DSS is a decision to upgrade the NAVY's F-14 Fleet Interceptor. In this decision process, many decision variables referred to as **COMPONENTS** are considered for upgrade. Each **COMPONENT** has the possibility of being upgraded to one of several configurations called **OPTIONS**, each with differing costs and utility values to the overall upgrade configuration.

In this process, an additive combination of one **OPTION** per aircraft **COMPONENT** is chosen. The goal of the SCP-DSS is to find the optimum combination of **COMPONENT-OPTIONS** that returns the highest utility score.

A. SUMMARY DESCRIPTION

To determine the combination of **COMPONENT-OPTIONS** that returns the greatest utility for a given amount of money, the first task is to review all **OPTIONS** within each **COMPONENT** and assign a **UTILITY** value for each. Decision makers meet to give their opinions and validations to the building of the utility matrix. Accompanying the utility matrix is the cost matrix. The cost matrix is a utility matrix template copy containing costs for each option. All option costs are determined prior to the utility evaluation phase so that the matrices can both be built simultaneously. Each matrix provides input paths for the decision makers as they progress through the decision process.

A zero-to-100 percent scale is used to guide the decision makers through their judgments on the relative merits of all **COMPONENT** options. The current aircraft configuration is assigned the value of zero percent. The "ideal" option is assigned the value of 100%. All other options are scored by relative importance between these extremes. That

is, the ENGINE has seven options under consideration; each option is scaled and scored relative to only the other six options within the ENGINE component. Measures of UTILITY are shown in their resultant ranking in Figure 3-1.

COMPONENTS	OPTIONS						
ENGINE	0	2	10	12	13	15	100
RADAR	0	3	30	40	60	100	
ELEC-OPTC	0	3	45	90	100		
TAC COMMS	0	0	20	30	65	100	
AVIONICS	0	3	9	10	90	95	100
EW	0	50	55	60	100		
WEAPON INT	0	40	70	85	100		
WPN CONT	0	50	80	95	100		
SURVIVE	0	45	70	90	100		
WT REDUC	0	34	59	64	85	100	
OSIP	0	20	40	40	60	100	

Figure 3-1. Utility Values of Options (%)

Figure 3-2. shows option cost figures that template the COMPONENT-OPTION configuration as they appear relative to one another in UTILITY value. Note that the costs of the options do not affect the relative utility relationship of the options to one another. The decision makers consider the utility value of the COMPONENT-OPTIONs as the most critical decision element rather than the cost of each.

COMPONENTS	OPTIONS						
ENGINE	0	2	0	1	2	1	353
RADAR	0	0	0	162	202	133	
ELEC-OPTC	0	5	20	70	65		
TAC COMMS	0	0	0	0	40	20	
AVIONICS	0	20	20	20	380	40	40
EW	0	10	25	0	15		
WEAPON INT	0	20	3	20	30		
WPN CONT	0	0	15	15	18		
SURVIVE	0	3	9	4	4		
WT REDUC	0	0	10	20	60	200	
OSIP	0	0	0	0	15	18	

Figure 3.2. Specific Costs of Options (\$Million)

The next task is to determine a ranking of the desirability between the COMPONENTS. In this step, the relative importance, or priority, of the components is agreed upon. Since a base percentage of desirability could not be readily determined, as in the ranking of options, the decision makers now assigned arbitrarily relative weights based on the overall importance of each COMPONENT. Raw numbers are used to aid in the simplicity of scaling the importance of one COMPONENT to another. Summing the COMPONENT relative weights and using this sum to divide into the relative scores normalizes the entire matrix to a 100 percent scale. This normalizing process reduces the complexity of the final solution. This ranking is shown in Figure 3-3.

These raw weights are then translated into percentages as shown by Figure 3-4. The raw relative weight for the ENGINE of 210 divided by the summation of all weights—637—yields the percent value of 32.97 percent.

COMPONENTS	
210	ENGINE
100	RADAR
50	ELEC-OPTC
50	TAC COMMS
80	AVIONICS
40	EW
35	WEAPON INT
25	WPN CONT
12	SURVIVE
25	WT REDUC
10	OSIP
TOTAL	637

Figure 3.3. Relative Desirability of Components Using Raw Weights

COMPONENTS	%
210	32.97%
100	15.70%
50	7.85%
50	7.85%
80	12.56%
40	6.28%
35	5.49%
25	3.92%
12	1.88%
25	3.92%
10	1.57%
TOTAL	637

Figure 3.4. Component Relative Weights as a Percentage

The data on option utility, costs , and relative importance of each COMPONENT is the input to the SCP-DSS.

With manual or mainframe computer methods, based upon locality to the decision makers, the decision result time is measured in days. The process took days because the

data had to be collated, then taken to the location of the mainframe computer, entered into the machine, results tabulated, and these results returned to the decision makers for a final decision to be made. Decision makers wanted more timely feedback than currently available. The entire process as described in this scenario using a microcomputer-based SCP-DSS will decrease time requirements to minutes. Manual and mainframe processes are too laborious to use. Answers to numerous "what if" suppositions can be given without excessive delay due to the locality of the decision makers and the availability of the SCP-DSS.

B. PRELIMINARY CALCULATIONS

Prior to final calculations which yield the optimal combination, a Cumulative Costs matrix is produced. The Cumulative Costs matrix single option cost includes the costs of the options which precede it. Figure 3-5 shows the Cumulative Costs matrix of the example model. The matrix is derived using the cost data shown by Figure 3-2.

COMPONENTS	OPTIONS						
	0	2	2	3	5	6	
ENGINE	0	2	2	162	364	497	
RADAR	0	0	0				
ELEC-OPTC	0	5	25	95	160		
TAC COMMS	0	0	0	0	40	60	
AVIONICS	0	20	40	60	440	480	520
EW	0	10	35	35	50		
WEAPON INT	0	20	23	43	73		
WPN CONT	0	0	15	30	48		
SURVIVE	0	3	12	16	20		
WT REDUC	0	0	10	30	90	290	
OSIP	0	0	0	0	15	33	

Figure 3.5. Cumulative Costs Matrix

The SCP-DSS calculates the Weighted Criteria Utility matrix. The objective of these calculations is to combine the utility values of options and COMPONENTs into a single matrix of weighted utility values. To do this, the relative percentage weight of each COMPONENT is multiplied by the utility score for each option. Figure 3-6 shows the completed calculations in the weighted utility matrix. Note, for example, that the calculation for RADAR-Option 5 of 9.42 is yielded by multiplying 15.7 percent (as shown in Figure 3-4) by the option score of 60 percent (as shown by Figure 3-1).

	COMPONENTS	OPTIONS						
32.97%	ENGINE	0	0.66	3.3	3.96	4.29	4.95	33
15.70%	RADAR	0	0.47	4.71	6.28	9.42	15.7	
7.85%	ELEC-OPTC	0	0.24	3.53	7.06	7.85		
7.85%	TAC COMMS	0	0	1.57	2.35	5.1	7.85	
12.56%	AVIONICS	0	0.38	1.13	1.26	11.3	11.9	12.6
6.28%	EW	0	3.14	3.45	3.77	6.28		
5.49%	WEAPON INT	0	2.2	3.85	4.67	5.49		
3.92%	WPN CONT	0	1.96	3.14	3.73	3.92		
1.88%	SURVIVE	0	0.85	1.32	1.7	1.88		
3.92%	WT REDUC	0	1.33	2.32	2.51	3.34	3.92	
1.57%	OSIP	0	0.31	0.63	0.63	0.94	1.57	
100%								

Figure 3.6. Weighted Utility Matrix

C. BUILDING THE UTILITY CURVE

With the Cumulative Costs and Weighted Utility matrices, the main process of determining the best combination of options begins. Initially, the decision makers chose funds (cost) as the parameter limit to drive the SCP-DSS to the optimum combination of options. In this example, the funds (cost) are the limiting parameter.

The SCP-DSS generates a utility curve from data inputs. This aircraft upgrade had the cost possibility range of zero to \$2.1 billion. Twenty equal intervals of SCP-DSS generated increments are used as predetermined steps to guide the plotting function (i.e., 2.1 billion divided by 20 gives increments of 105 million). The objective of this plotting function is to give the user a visualization of his data matrices and whether it will produce a risk curve plot similar to one of the types shown in Figure 3-7. Note that if a utility curve can be constructed, then one can select utility values that correspond to any desired monetary value. The construction of the curve, therefore, is key to the analysis. According to the Von Neumann-Morgenstern proposal [Ref. 13], a curve can be constructed by measuring the attitude of the decision maker toward risk. This measure assesses the risk in monetary

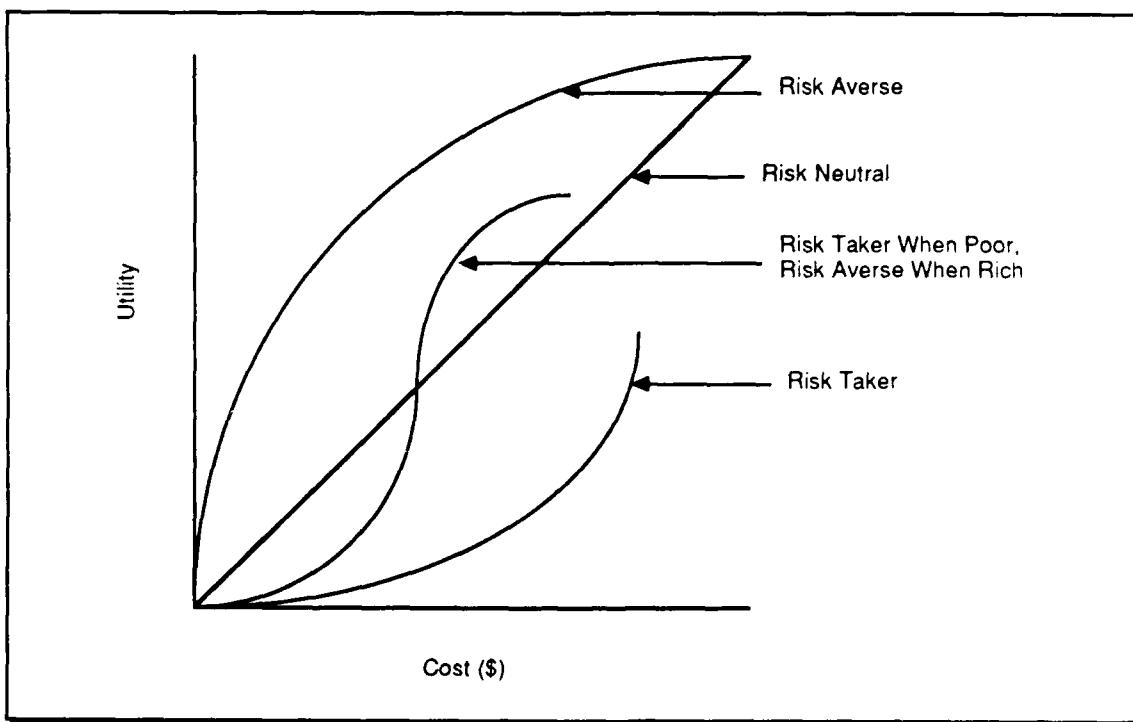


Figure 3.7. Utility Curves

amounts that the user is willing to lose as compared to the expected return in utility for the money the risk taker spends. The shape of the curve is a function of the decision maker's attitude toward risk. The conservative perspective of the DOD decision maker is that of a risk-averse decision maker. Using the risk averse curve, for example, money has a lot of value, incrementally per dollar spent, when a risk averse decision maker is poor. However, beyond a certain amount, the monetary increases have less and less value, incrementally, as the amount of money increases. Once a decision maker's utility curve is known, then it is possible to replace any monetary value by its utility equivalent for that decision maker.[Ref. 4]

Figure 3-8 shows a utility curve of 20 equal steps of \$105 million each. An algorithmic plotting procedure within the SCP-DSS produces a utility curve for the user.

The plotted curve is analyzed to determine the theoretical point of the highest utility score per cost value given. This is the point called the Decision Point (DP). The DP is located on the outermost frontier of the utility curve at the intercept of a tangent line to the curve having a risk neutral utility curve.

Marginal cost analysis theory is used to assist in the verification of the highest utility score at a tangent point along the plotted curve parallel to the risk neutral curve (see Appendix B for references on Microeconomic Marginal Theory). All slope functions prior to this DP will have a slope-value greater than one, meaning that there is incrementally more utility per dollar spent, whereas after the DP the slope-value along the curve approaches zero, indicating the utility increment per dollar spent is smaller and smaller as the curve slope-value approaches zero. In this example the DP has a 71 percent utility value and a 41 percent cost value (approximating \$800 million).

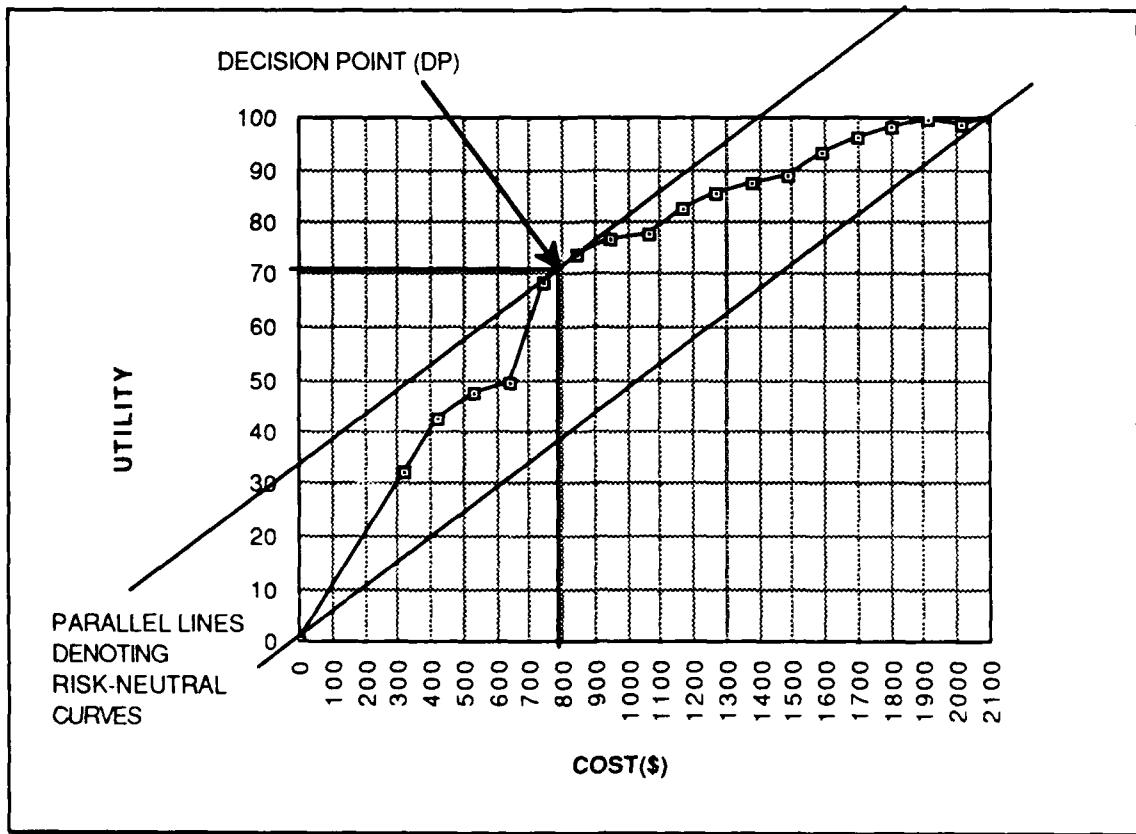


Figure 3.8. Utility Curve Plot Showing Decision Point

D. COSTS RANGE CALCULATIONS AND RESULTS

The decision makers can now determine a cost range within which to consider options. This cost range may be the theoretical decision point (DP) of the Highest Utility, an imposed spending ceiling, a minimum desired utility, or any possible combination of economic and/or political considerations.

The decision makers use the \$800 million cost as the input for the SCP-DSS to determine the optimal combination of options. The program offers the user single cost targets or cost targets with a range (i.e., \$800 million plus or minus \$2 million). The SCP-DSS will consider all combinations within the range as possible answers for comparison;

however, it only gives the decision makers the highest utility combination within the identified range. The computed output can be modified by altering the input utility scores or altering the relative **COMPONENT** weights. The overall matrix shape can be changed to project certain minimum and maximum considerations in option choices for the decision makers. The SCP-DSS in its prototype configuration can be easily modified to allow the decision makers to tailor the system to adapt to the possible unique criteria that they may choose.

IV. THE SCP-DSS SYSTEM DESIGN

The objective of system design is to develop a blueprint for the physical system from the analysis output. The system view changes direction from "what" to "how." A prototype system design serves as a model or framework for the final system. The system plan must be a complete system design, not a partial design. The prototype becomes an extension of the feasibility study; its purpose is to demonstrate feasibility. The functional requirements for the data component, the dialog component, and the model component are inputs to the prototyping phase. The design of the prototype is key to its successful and rapid completion. User-oriented design helps build a DSS while promoting a shorter development. Better problem refinement will result in greater user satisfaction in the end [Ref. 14].

A. USER-ORIENTED DESIGN

User-oriented design has three major components:

- User-controlled systems design
- User-defined criteria of system quality
- Special attention to design of the interface between user and system

User control is the most important element of user-oriented design. The user's influence on the final system is the result of close cooperation between the user and designer. Interviews helped the designer to construct a system by translating user needs into the technical specifications for computer programmers. The user helps to determine his own trade-offs as to the system inputs and outputs that are considered necessary for the decision process. Cooperative design ensures system quality which is measured by system usability and efficient error processing.

The final component of user-oriented design is the interface between the user and the system. The essence of this DSS design is to structure a user-friendly dialog component on a microcomputer, so that decision makers can easily input their data, then process their matrices in a timely fashion in a local and secure manner. Mason alludes to the value of a dialog component by saying that the closer the information provided is to the decision-maker's needs, the better the decisions that will be made [Ref. 15]. This means design of this DSS must stay close to decision-makers' concepts of their decision processes while assisting them toward a final decision. Local use of microcomputer resources at the decision-makers' disposal is key to supporting all phases of the decision-making processes. Evaluating options and weighting components enjoins experts to evaluate the result each option score and component weight has on the final decision. The matrix is an outline to support the semistructured nature of this type of decision. An adequate interface between the user and the system ensures a higher quality input and output. Without a proper and usable interface, the user loses confidence in the system.

1. User-Oriented Analysis

The user-oriented analysis process allows the following:

- Users learning about the decision domain and tasks required.
- Specifying the performance criteria by giving users reassurance that the system will give similar answers every time given similar input data.
- Selecting a DSS building tool (if available) to consider type of modelling component to assure support to a decision (PASCAL was chosen).
- Developing an initial implementation to review use of the model and allow users to become more committed to the DSS.
- Testing implementation to determine the viability of continuance of the project. This was accomplished via demonstration to the users.
- Developing detailed design for a complete system.

2. User-Oriented Implementation

When everyone is satisfied that the prototype can perform as designed, the implementation development follows. Implementation includes the following activities:

- Implementing the system structure from logical design through physical implementation.
- Tailoring user interfaces to make it easy for the user to input data and receive output, query the system, and add, delete, or change existing data.
- Monitoring system performance.

During the implementation phase, components are tested against performance and decision criteria obtained during earlier stages of the prototype development. Alternate data inputs are used for similar-style problems to see if the dialog component is standard for new data inputs and that the model component is satisfactory for the problem solution set. Testing continues to refine the dialog component and adapt to newer and better ways to input the data component. If newer methods and modelling algorithms are discovered to be more elegant for solution sets, then they can be added to the system.

B. FUNCTIONS OF THE SCP-DSS

The SCP-DSS should perform the following functions:

- Support decision makers by evaluating alternatives and choosing combinations of component-options that provide the highest total utility score based upon initial inputs.
- Provide graphical output, tabular analyses of data, and report formats to help decision makers select and tailor alternatives of choice.
- Provide for "what-if" analysis by being able to selectively change data inputs.

C. DATA FLOW OF THE SCP-DSS

The overall design of the SCP-DSS system is similar to the existing method except that the computation is local to the user in the microcomputer environment. The search for alternative solutions directs the designer to consider the most efficient microcomputer

automation techniques. Implementation strategies are reviewed to determine all system automation boundaries and the areas of input data processing. System decomposition determines the extent of duplicate processes and data flow paths. Figure 4.1 shows the highest level data flow diagram of the SCP-DSS. A logical decomposition model is presented in Appendix C for each process and data flow path.

D. THE DATA COMPONENT

The data component consists of the data that the SCP-DSS must process for the desired output information. The purpose of the DSS is to generate the desired combination of component-options with the highest utility score. The input begins with the user defining the matrix array size by inputting the number of rows (# ROWS) and the number of options per row (# OPTIONS PER ROW). The user is allowed to name the components in a character string of 15 characters maximum. The size of the matrix limits the user only to the number of data elements input, not the data element size. Numerical data types are integer or real values. The matrix array size is set for the cost (COST PER OPTION), utility (UTILITY PER OPTION), and weighted criteria (WEIGHTED CRITERIA PER ROW) inputs. All data inputs are stored in dynamic memory. The last user data input is the target cost (TARGET COST) and search range (SEARCH RANGE) to the CALC COMBO procedure. The CALC COMBO procedure calculates the combinational utility score using a TARGET COST incorporating the Cumulative Costs matrix and the Weighted Utility matrix. The CALC UTILITY CURVE procedure calculates the combinational utility score for the 20 target cost increments using the same two matrices. The highest utility score with its accompanying cost is stored in an array and passed to the PLOT CURVE procedure that outputs a utility curve for the user.

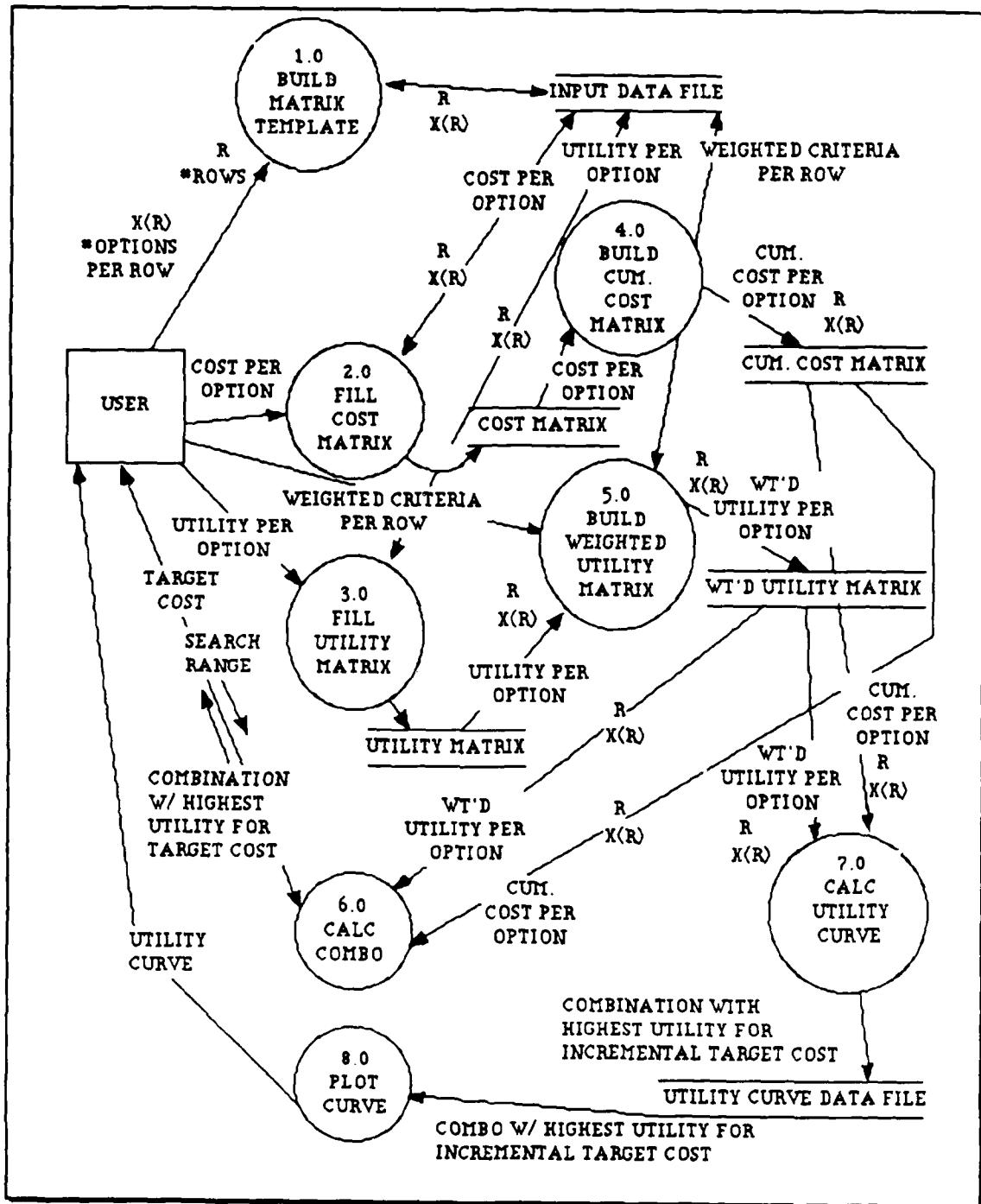


Figure 4-1. SCP-DSS Data Flow Diagram

Data inputs in dynamic memory may be modified at any time. Users may save data inputs before and/or after processing. To modify a saved set of data requires that the saved data be recalled from peripheral memory; only then can the cost, utility, and weighted criteria data be modified. Recalled data is loaded into the dynamic array structures for further processing. The arrays are passed between procedures within the program as needed.

The data component needs the flexibility to deal with a library of problems. To achieve this, the SCP-DSS must have the ability to save and retrieve files named by the user.

The user follows the following algorithm:

- The SCP-DSS queries the user to create a new data set or retrieve an already saved file.
- If new data is selected, then
 - the system queries the user for row, option , costs, utility, and component weights to build a data set for decision support

Else

- the system queries the user for the saved data set file name and begin to process the old data
- When the user is finished with the data set, the SCP-DSS queries the user to save or discard the current data set
- If save, then
 - save the data set in a file named by the user for later use

Else

- terminate program.

E. DIALOG COMPONENT

The dialog component must be the most elegantly designed component of the SCP-DSS. The format must be obvious for the user to balance user requirements against DSS function. The dialog component should guide the user through the data input and retrieval

with a minimum of effort. The SCP-DSS Dialog component consists of the following functions:

- user interface
- component linkage

1. User Interface

The designer's primary concern for the user interface is to make the SCP-DSS "user friendly." The power in the model component will not be used properly if the user interface is unacceptable. The screen frame should be uniform from beginning to end. This ensures the user that he is working in the same environment. Using a standard screen reduces the learning curve and increases acceptability.

The machine interaction should include menus, queries, error messages, standard input/output windows, graphic output, and a help facility.

a. Standard Screen

The standard screen area is divided into three areas: the working area, the menu area, and the message area.

- Working Area—In this area, the user can view the matrices and the combination answer. Any graphics processing is viewed in this area.
- Menu Area—In this area, the menu selections are available for the user to select the input, view, calculate, or perform graphics options as desired. Figure 4.2 shows the menu hierarchy structure.
- Message Area—This area displays alerts and input error messages to guide the user to input the correct data elements for the SCP-DSS. Alert and error messages can appear as dialog boxes. Whenever an error occurs or the SCP-DSS needs more information from the user, it presents a dialog box on screen. Dialog boxes are not for data input; they alert the user to an exceptional condition.

b. Input/Output

The input for this SCP-DSS is received from the keyboard or as a retrieved file from memory. The output is to the computer screen, printer, or saved file

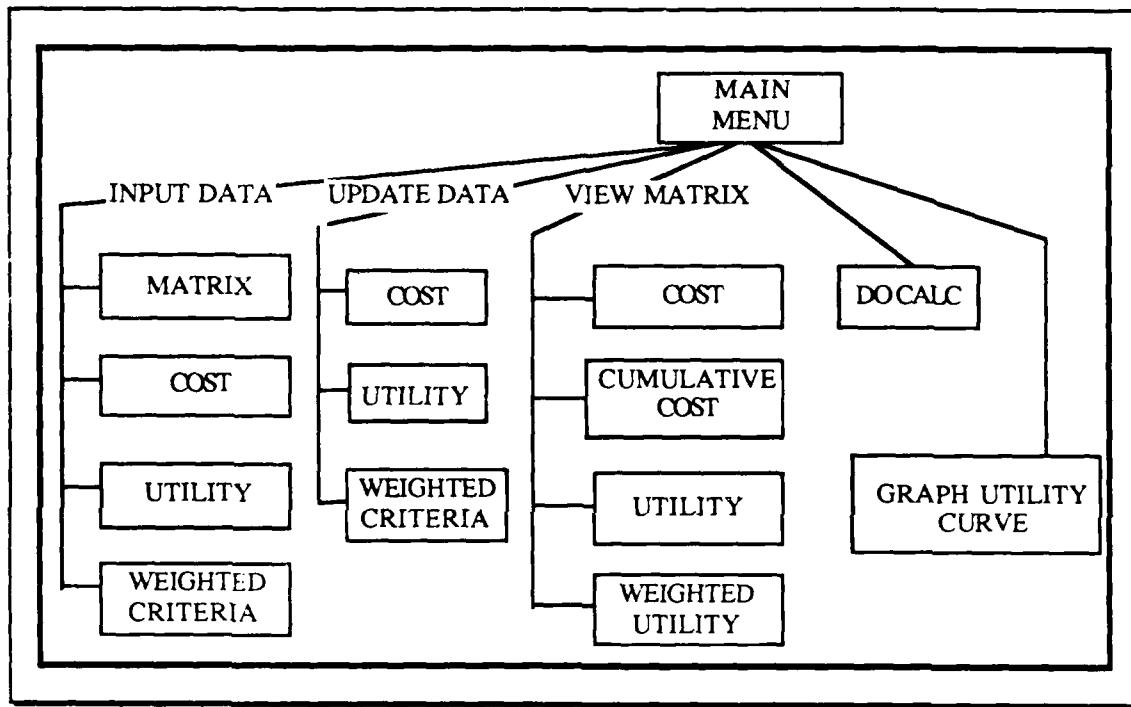


Figure 4.2. SCP-DSS Menu Hierarchy

format. The SCP-DSS output to printer gives the user backup when data is not saved to a file. The linear format plot is the most appropriate for this application because of its similarity to the Von Neumann-Morgenstern utility curve representations. The graphics output helps the user to conceptualize the differences among the alternatives present for analysis. The analysis of the graphics output helps identify key value areas from which to use the SCP-DSS to retrieve detailed information.

c. *Reports*

Printed reports are not a part of the SCP-DSS; however, this function should be added to give the user a summary of the processed data and sensitivity analysis. This summary should consist of the input data in both tabular and matrix form. The sensitivity analysis report should consist of the combination of options answer as well as the current changes made to the input data.

d. Help

The intent of the help function is to provide the user with on-line assistance and information about the specific area of the SCP-DSS currently being used. Each help screen is written to an appropriate level of detail and is presented on a single screen.

2. Component Linkage

This ingredient of the user interface assures the connections of model components with data components. The use of IF_THEN_ELSE or CASE type statements is appropriate for functions or procedure control. PASCAL offers arrays as a structured type variable whereby each array can contain distinct but related components. The array allows for data grouping and indexing by element type. The array structure is highly dependent on the programming language and the hardware configuration used for the SCP-DSS.

F. SCP-DSS MODEL COMPONENT

The most important functions of the model component are the matrix building, model execution, dialog interface, and data interface functions.

1. Matrix Building Functions

Matrix building functions are required as building blocks to the model execution unit. These matrices form the content base for the search to determine the optimum combination of cost and utility options.(see Appendices D and E for source code listing).

a. Cost Matrix

This function fills the cost matrix by placing all input cost elements in the matrix designated by the number of rows, R , and the number of option per row, X_R . The cost elements are assigned to their respective component options as structured by the matrix array.

b. Utility Matrix

This function fills the utility matrix by placing all input utility score elements in the matrix designated by the number of rows, R, and the number of option per row, X_R . The utility score elements are assigned to their respective component options as structured by the matrix array.

c. Cumulative Costs Matrix

This function computes the cumulative costs matrix by summation of all input cost elements prior to its position in the matrix. The cost elements are assigned to their respective component options as input to the cost matrix. The summation equation is as follows for each component-option cell in the cumulative costs matrix:

$$\sum \text{comp}(1 \text{ to } X_R) = \text{comp}(1_R) + \text{comp}(2_R) + \dots + \text{comp}(X_R)$$

where X_R is the number of component-options per row R.

For example, the cumulative costs for component-option number 4 is the sum of input costs for components numbered 1 through 4 inclusively (see Figures 3.2 and 3.4 in Chapter III).

d. Weighted Utility Matrix

This function computes the Weighted Utility matrix by summing all the component variable weights and dividing the input weights by the total (see Figure 3-5 in Chapter III). The Utility elements are assigned to their respective component options as input for the Utility matrix and then multiplied by the calculated percentage value.(see Figure 3-6 in Chapter III).

2. Model Execution

The model execution function calculates the combination of options with the highest utility score for a given input target cost. The intent is to pick one option per row

and add it to the next row's option choice. The algorithm recursively loops through all component-options in each row, scores each set's utility score, and saves the utility score and the combination, if it is higher than the last saved score and combination. The unit checks all combinations, in order, to show the user that all possible choices have been evaluated. The thoroughness of this unit is critical because if a partial set of combinations are evaluated then only partial decision effectiveness is gained.

The following pseudocode details the execution algorithm:

ROW 1 has 3 options

ROW 2 has 4 options

ROW 3 has 5 options

```
FOR X= 1 TO 3 DO
    FOR Y = 1 TO 4 DO
        FOR Z = 1 TO 5 DO

        COST = COST(X) + COST(Y) + COST(Z)
        UTILITY= WT'D UTILITY(X) + WT'D UTILITY(Y) + WT'D UTILITY(Z)
        IF UTILITY> HIGHEST_UTILITY THEN
            HIGHEST_UTILITY = UTILITY
            X1 = X
            Y1 = Y
            Z1 = Z
        NEXT Z
    NEXT Y
NEXT X
```

3. Dialog Interface

The model component is directly interfaced with the dialog component so that the user can direct the matrix building and execution phases. The user selects the desired target cost to drive the execution unit to search for the optimum utility score and

component-option combination. The user determines whether to derive the utility curve for the entire problem or to immediately search the combination answer.

4 . Data Interface

The model component is directly interfaced to the data component. The model component begins by accepting the user input and then giving the option for saving the data and for data retrieval. Files can be created or deleted at the user option. Figure 4-3 shows the data flow in the system user interface from user input to SCP-DSS output.

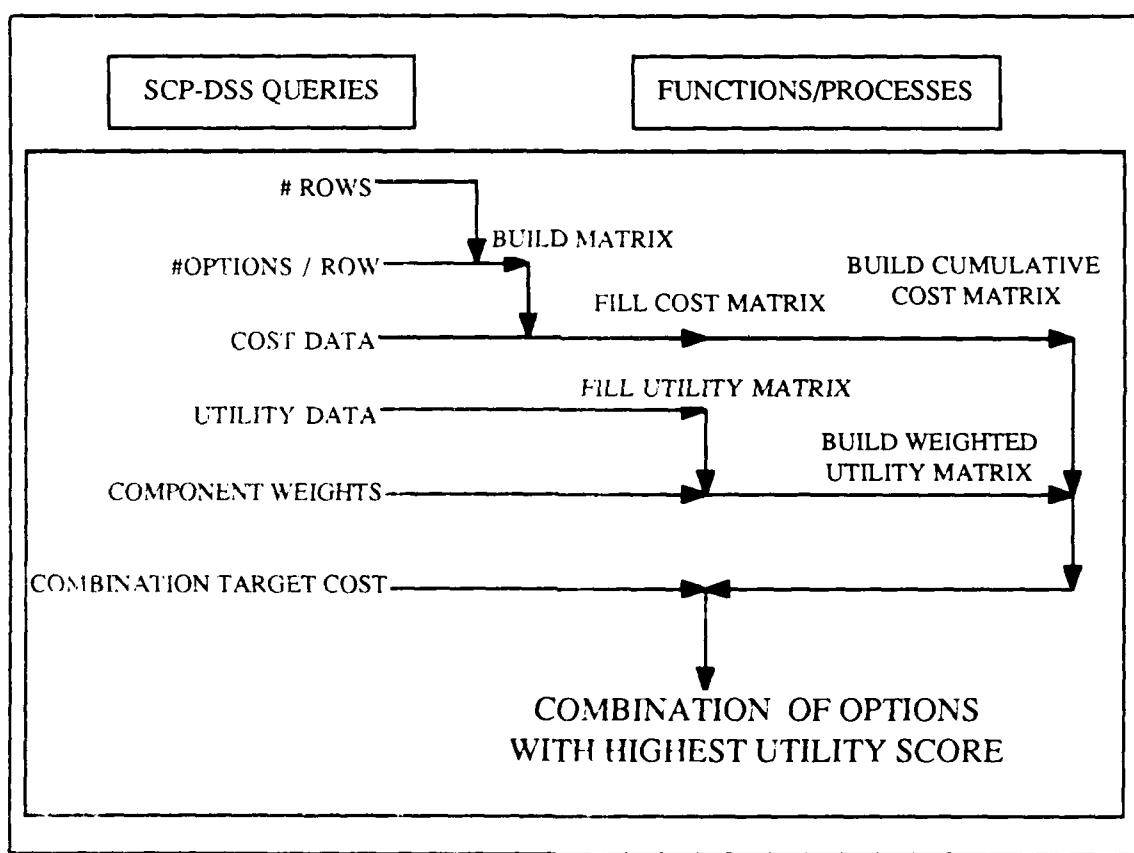


Figure 4-3. SCP-DSS User Inputs to System Output

V. PROTOTYPE IMPLEMENTATION OF THE SCP-DSS

The process of prototyping allows concurrent evolution of user requirements and system design [Ref. 16]. The following steps were used in the prototype process:

1. Users' basic requirements were identified by interviews and user feedback to gain the information needs and decision support requirements.
2. Incremental development of a working prototype that performs all important, identified functions, using sample representative input data.
3. Allow the user to test the prototype and evaluate its performance and output for update and immediate modification.
4. Further refine the prototype by discussing with the user the requested changes and deciding which ones were feasible for implementation. Repeat steps 3 and 4 until the system fully achieves the requirements of the users.

Prototyping as a system development and design methodology recognizes cognitive-style issues, requires advanced technology (applying PASCAL programming to the micro-computer) and is an adaptive revision to accepted systems development methodology [Ref. 17]. The goal is to develop a working system that is refined through an iterative process with major user involvement. Figure 5-1 represents the requirements determination and validation by the prototyping process [Ref. 18]. Many times, prototypes are, at some point in development, discarded and a formal system development process is initiated. Alternatively, the prototype may become the production system.

The prototype process, including editing and updating, is used successfully here with the following benefits:

- Shorter development time
- Better problem definition and refinement
- Greater user participation and support
- Greater user satisfaction

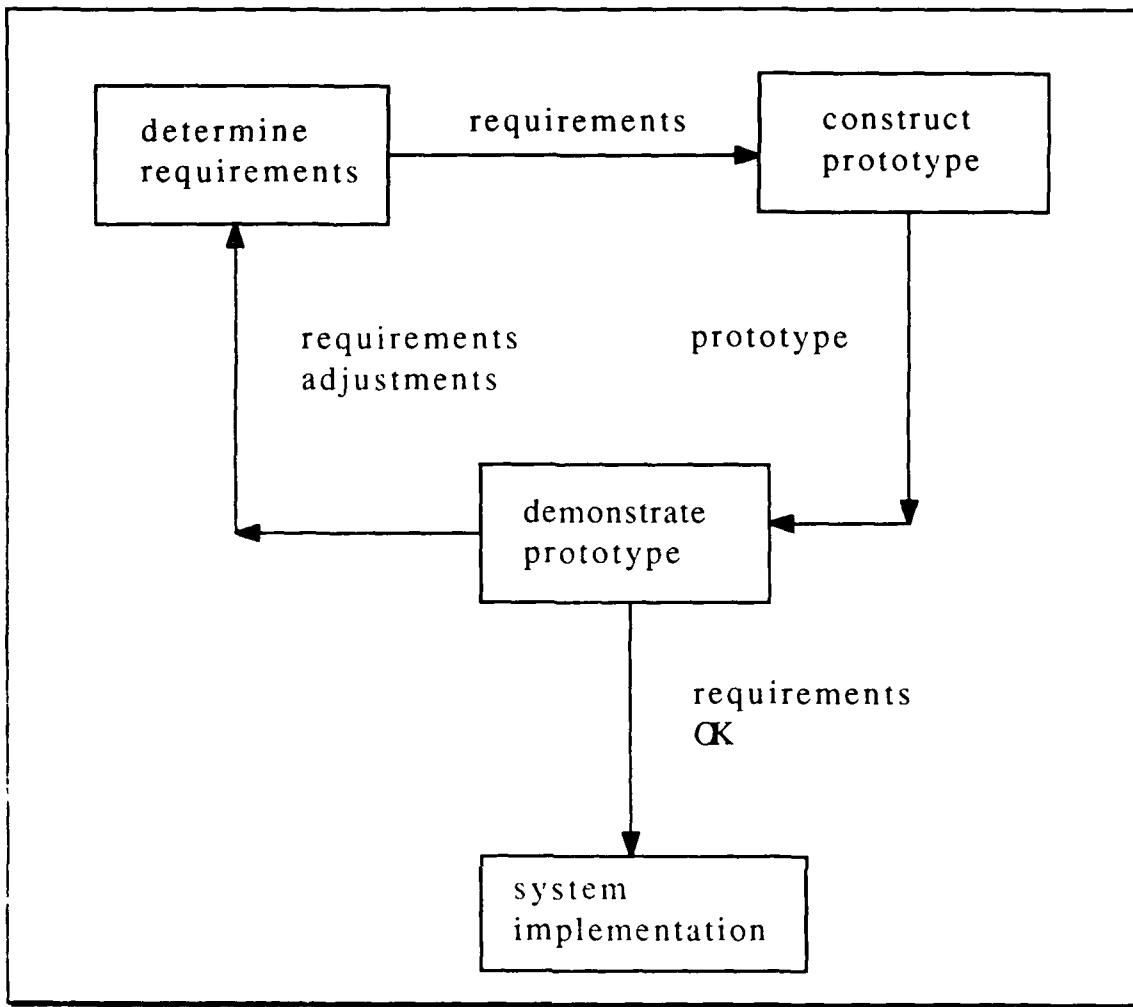


Figure 5-1. Process of Requirements Determination
and Validation by Prototyping

The development of many systems is facilitated using a standard System Development Life Cycle (SDLC) approach. When the user requirements are unclear or too broad in scope for standard methods, then prototyping becomes an acceptable alternative to SDLC. Prototyping offers to the design process an articulate method of quick feedback and refinement to determine user requirements and get them automated as soon as possible.

A. PROTOTYPING LANGUAGE

The language used for prototyping this problem is PASCAL. Pascal was developed in 1971 by Nicklaus Wirth. Its syntax is relatively easy to learn and its structured nature supports programming that is easy to read, understand, and maintain [Ref. 19]. Pascal as a high-level language is much easier to use than machine or assembly languages. The intent is to write code that is portable so that it can be executed without modification on many different types of microcomputers. An assembly or machine-language may only execute on one computer.

B. PROTOTYPING PROBLEMS

1. User Requirements

User requirements for this problem were gained by interviews and discussion. From these meetings, the following areas of importance to the solution were gained:

1. The nature of the in-house systems command decision environment.
2. The local automation of the data analysis.
3. The decision time element whereby, if it were shortened to minutes from days, a more thorough study of the information results could be gained.
4. Integer input data types and formatted real values in the data manipulation.
5. The desired matrix format and plotted view of the desired output.
6. Analytical insights into user's decision algorithms and processes.
7. All possible combinations are to be compared to ensure thoroughness of calculation.

User involvement was critical for initial momentum and insights. The determination of the decision environment structure in which this decision tool will be used led to setting the level of user-to-system interaction necessary. Also, the degree to which data analysis can be performed to assist the users in making their decisions. Above all, this system had to be user friendly, fast, and thorough in computation.

2. Programming Environment

A major problem in implementing this prototype in the microcomputer environment was the magnitude of calculations required with the relatively slow speed of the microcomputers used. Results were gained using a set of generic sample data for the F-14 upgrade process described in Chapter III. Due to the requirement that all possible combinations be compared during each trial run, note in Figure 5-2 the times recorded for the matrix sizes.

		ROWS						
		1	2	3	4	5	6	7
C	1	1	1	1	1	1	1	1
	2	2	4	8	16	32	64	128
	3	3	9	27	81	243	729	2187
	4	4	16	64	256	1024	4096	16384
	5	5	25	125	625	3125	15625	78125
	6	6	36	216	1296	7776	46656	279936
	7	7	49	343	2401	16807	117649	823543
	8	8	64	512	4096	32768	262144	2097152
	9	9	81	729	6561	59049	531441	4782969

NUMBER OF CALCULATIONS RELATED TO MATRIX SIZE

		ROWS						
		1	2	3	4	5	6	7
C	1	0.004	0.004	0.004	0.004	0.004	0.004	0.004
	2	0.008	0.016	0.032	0.064	0.128	0.256	0.512
	3	0.012	0.036	0.108	0.324	0.972	2.916	8.748
	4	0.016	0.064	0.256	1.024	4.096	16.384	65.536
	5	0.02	0.1	0.5	2.5	12.5	62.5	312.5
	6	0.024	0.144	0.864	5.184	31.104	186.62	1119.74
	7	0.028	0.196	1.372	9.604	67.228	470.6	3294.17
	8	0.032	0.256	2.048	16.384	131.07	1048.6	8388.61
	9	0.036	0.324	2.916	26.244	236.2	2125.8	19131.9

ALL TIMES IN SECONDS

Figure 5-2. Time Data Related To Matrix Size

The magnitude of calculations that are required and the time involved for a solution to the problem set varied from 4 to 11 minutes, depending on the computer used. Data sets up to seven by seven matrices sizes were measured. Larger matrix size time estimates are extrapolated from the sample data trials, given that computer processor speeds are constant.

In writing the program (Source Code listings are in Appendices D and E), several steps were used to reduce the computer run time. The first step was to limit the input/output operations. This was accomplished by not displaying to the screen each combination as it was computed. Limiting input/output operations made the process CPU intensive. This resulted in more processor time being devoted to actual calculations.

A second step was to restrict the allowed input/output to main memory. Because dynamic memory access is faster than peripheral access, the CPU exercised more directly on calculations than stopping for disk access to get added data.

The third step was to limit the number of possible combinations of utilities being considered by removing the zero cost cells from the combination sets. This requires a common sense understanding of the input matrices. Only the matrix cells with associated cost values are considered. This step is based on cost elements only, so the zero cost cells had to be present for relative importance of decision variables but not calculations. This reduced the matrix size for calculation. This reduction assumed that options with utility but no cost would automatically be chosen.

VI. CONCLUSIONS AND RECOMMENDATIONS FOR FURTHER STUDY

A. CONCLUSIONS

1. The micro-computer based SCP-DSS provides a mathematically unbiased result to decision makers involved in costs planning for major systems. Political elements and biases are either eliminated or taken into consideration by the experts in evaluating the utility of each option of each component and then assigning relative weighting values. The relative weighting of components accommodates the view that item costs do not determine utility value. The association is with the quality of the component's option in relation to comparative options.
2. Decision makers using this tool can judge the sensitivity of the data input given the results in both tabular and graphic form. In essence, more timely, precise decisions may result from the speed, and accuracy, with which this process can return information to the user.
3. The use of a microcomputer with graphics capability and user friendliness leads to a more conducive prototyping/programming environment. The PASCAL code facilitated a structured and simplified modular development on the Macintosh™. Integration of this code to the IBM PC™ or Zenith systems prevalent in the Department of Defense will require slight modifications of the current system. The data types and array structures are common in all PASCAL environments. The section of code used for plotting the utility curve will have to be removed or modified to operate on other than Macintosh™ microcomputer operating systems. Another concern for modifying this code to other machines involves the file manager routines necessary for saving data for later recall and use. This, again, is due to the nature of the operating system.
4. Marginal cost analysis theory is used to verify the highest utility score at a tangent point along the plotted curve parallel to the risk neutral curve. All tangent points to the utility curve prior to the decision point (DP) will have a slope-value greater than the slope value of a risk-neutral utility curve, meaning that there is incrementally more utility per dollar spent, whereas after the decision point (DP), the slope-value along the curve approaches zero, indicating that the utility increment per dollar spent is smaller and smaller as the curve slope-value approaches zero.
5. A prototype strategy for review of each component supports the development, design, implementation, and maintenance of this DSS. With its array structure and size, based on user input, the data, dialog, and model components are easily separated and maintained. The data component is simplified by use of array structure types. The model component, although not intricate in design, becomes complex when tied to a dialog component that is a complex, and restrictive, user interface. The dialog component is dependent on the user friendliness of the coded program or the microcomputer operating system environment. The model component is driven

by the complexity of the data component via the dialog component. This tool, designed and refined through prototype development, is automated so that data input is simple, changes to inputs can be made easily, and results can be calculated quickly.

6. The SCP-DSS can support later phases of decision and sensitivity analysis. This is accomplished by using the SCP-DSS as a tabulating system for the data inputs. Choices have to be equitably made. When the options become too numerous for simple decision making then quantifiably reducing the decision elements into groups is needed. If grouping is unacceptable, then the use of automation to consider each element for decision making becomes necessary. When all data inputs are complete, the locality of this tool allows for many ad hoc "what-if" queries. The time element for combination calculation is minuscule in relation to the flexibility that this system offers.

B. RECOMMENDATIONS FOR FURTHER STUDY

1. It is recommended that the recursive array processing algorithm be studied for further optimization. The narrowing of the solution set prior to the array processing and reducing the array size into small chunks for individual processing of each chunk in parallel may help to reduce the time element for calculation of a solution combination.
2. Consideration of programming this DSS in an object oriented code such as ADA, the accepted DOD programming code standard, may facilitate this tool's proliferation and use.
3. This DSS should be used in evaluation of the impact on secure and classified decision making that can be made in the work environment.
4. This SCP-DSS should be evaluated on other planning applications like wargaming scenarios, tactics evaluation, and weapons deployment. The use of this DSS in parallel with an alternative method of both analysis and information results gathering for a decision environment may provide cross-over and alternative uses for this tool.
5. Investigation into the use of and redesign of this DSS in an interactive group decision environment. This tool, if properly implemented in a Group Decision Support System (GDSS), could further reduce the time of the entire decision process.

APPENDIX A

MULTI-ATTRIBUTE UTILITY THEORY

Multicriterion choice methods are directed at problems in which there is a finite set of predefined alternatives or choices. A widely known method for choice problems is Multi-Attribute Utility Theory (MAUT). The approach is to estimate the decision maker's value function (for deterministic problems) or utility function (for uncertainty situations) [Ref. 20]. The function, defined over the criteria, serves to collapse the problem into one with a single criterion, the maximization of utility. Once the utility function is known, solution identification is straight forward.

Multi-Attribute Utility Theory (MAUT) has been developed for problems which have uncertainty about outcomes (consequences). If an appropriate utility is assigned to each possible consequence and the expected utility of each alternative is calculated, then the best course of action is to take the alternative with the highest expected utility [Ref. 21]. MAUT is one of the more difficult topics under Multicriterion choice methods due to its sophisticated nature (assumption) and elaborate assessment of the utility function. Many advances in MAUT have been developed by Keeney [Refs. 22–26]. The literature on MAUT and its assessment methods has been summarized in Farquhar [Ref. 27], Fischer [Refs. 28–29], and Fishburn [Refs. 30–31]. Keeney and Raiffa [Ref. 32] in particular deal extensively with utility from unidimensional to multi-attribute, its assessment methods and applications.

Multi-Attribute Utility Theory (MAUT) is used for handling uncertainty in outcomes. Most of the literature is filled with mathematical proofs, but most of the theoretical work in MAUT investigates the possibilities for simplifying the task of MAUT assessment. Skepticism concerning the practical usefulness of MAUT involves the fact that MAUT has been

applied to relatively few applications. Many theorists have proposed a variety of models/methods describing how a decision maker might arrive at a preference judgment when choosing among multiple attribute alternatives. MAUT requires various types of underlying assumptions, information requirements from the decision maker, and evaluation principles; these include the following: complete independence among attributes, indifference curves for hierarchical tradeoffs, or—if situation dependent—maximin for pessimistic decisions, maximax for the optimistic, or disjunction for specialized selection [Ref. 21]. Decision making certainly is difficult; MAUT is one of several multicriterion choice methods.

Publications referenced in this appendix include:

Stadler, W., "Preference Optimality (On Optimality Concept in Multicriteria Problems)," in W. Oettli and K. Ritter (eds.), *Optimization and Operations Research*, pp. 129–306, Springer-Verlag, New York, 1976.

Hwang, C. and Yoon, K., "Multiple Attribute Decision Making, Methods and Applications," in M. Beckmann and H. P. Kunzi (eds.), *Lecture Notes in Economics and Mathematical Systems*, p. 208, Springer-Verlag, New York, 1981.

Keeney, R. L., "Quasi-Separable Utility Functions," *Naval Research Logistics Quarterly*, vol. 15, no. 4, pp. 551–556, 1968.

Keeney, R. L., "Utility Independence and Preferences for Multiattributed Consequences," *Operations Research*, vol. 19, no. 4, pp. 875–893, 1971.

Keeney, R. L., "Utility Functions for Multiattributed Consequences," *Management Science*, vol. 18, no. 5, part 1, pp. 276–287, 1972.

Keeney, R. L., "Concepts of Independence in Multiattribute Utility Theory," in J. Cochran and M. Zeleny (eds.), *Multiple Criteria Decision Making*, University of South Carolina Press, Columbia, South Carolina, 1973.

Keeney, R. L., "Multiplicative Utility Functions," *Operations Research*, vol. 22, no. 1, pp. 22–34 1974.

Farquhar, P. H., "A Survey of Multiattribute Utility Theory and Applications," in M. Starr and M. Zeleny (eds.), *Multiple Criteria Decision Making*, North Holland, New York, 1977.

Fischer, G. W., "Experimental Applications of Multi-Attribute Utility Models," in D. Wendt and C. Vlek (eds.), *Utility, Probability, and Human Decision Making*, D. Reidel Pub. Co., Boston, 1975.

Fischer, G. W., "Utility Models for Multiple Objective Decisions: Do They Accurately Represent Human Preferences?" *Decision Sciences*, vol. 10, no. 3, pp. 451-479, 1979.

Fishburn, P. C., "Lexicographic Orders, Utilities, and Decision Rules: A Survey," *Management Science*, vol. 20, no. 11, pp. 1442-1471, 1974.

Fishburn, P. C., "A Survey of Multiattribute/ Multicriterion Evaluation Theories," in S. Zionts (ed.), *Multiple Criteria Decision Making: Kyoto 1975*, Springer-Verlag, New York, 1976.

Keeney, R. L., and Raiffa, H., *Decision with Multiple Objectives: Preferences and Value Tradeoffs*, Wiley and Sons, New York, 1976.

APPENDIX B

REFERENCES ON MICROECONOMIC MARGINAL THEORY

The following reference list is not exhaustive but provides adequate discussion on Utility Theory and Analysis within the realm of Microeconomic Marginal Theory. The search for analysis summaries of a microeconomic approach on the Von Neumann-Morgenstern utility assessment curves are mentioned in each of these texts:

Henderson, J. M., and Quandt, R. C., *Microeconomic Theory: A Mathematical Approach*, New York: McGraw-Hill, 1958, ch.2.

Koplin, H. T., *Microeconomic Analysis: Welfare and Efficiency in Private and Public Sectors*, New York: Harper and Row, 1971, ch.3.

Koutsoyiannis, A., *Modern Microeconomics*, New York: Wiley and Sons, 1975, pp. 15-16.

Lipsey, R. G., and Steiner, P. O., *Economics*, New York: Harper and Row, 1969, ch. 11.

Lloyd, C., *Microeconomic Analysis*, Homewood, IL: Richard D. Irwin, Inc., 1969, pp. 35-65.

Luce, R. D., and Raiffa, H., *Games and Decisions*, New York: McGraw-Hill, 1970, pp. 21-22.

Malinvaud, E., *Lectures on Microeconomic Theory*, New York: North-Holland Publishing Co., 1972, pp. 16-20.

Nicholson, W., *Microeconomic Theory: Basic Principles and Extensions*, 2nd ed., Hinsdale, IL: The Dryden Press, 1978, pp. 57-60.

Rader, T., *Theory of Microeconomics*, New York: Academic Press, 1972, ch. 6.

Samuelson, P. A., *Foundations of Economic Analysis*, Cambridge, MA: Harvard University Press, 1947, ch. 5.

Shone, R., *Microeconomics: A Modern Treatment*, New York: Academic Press, 1975, pp.59-60.

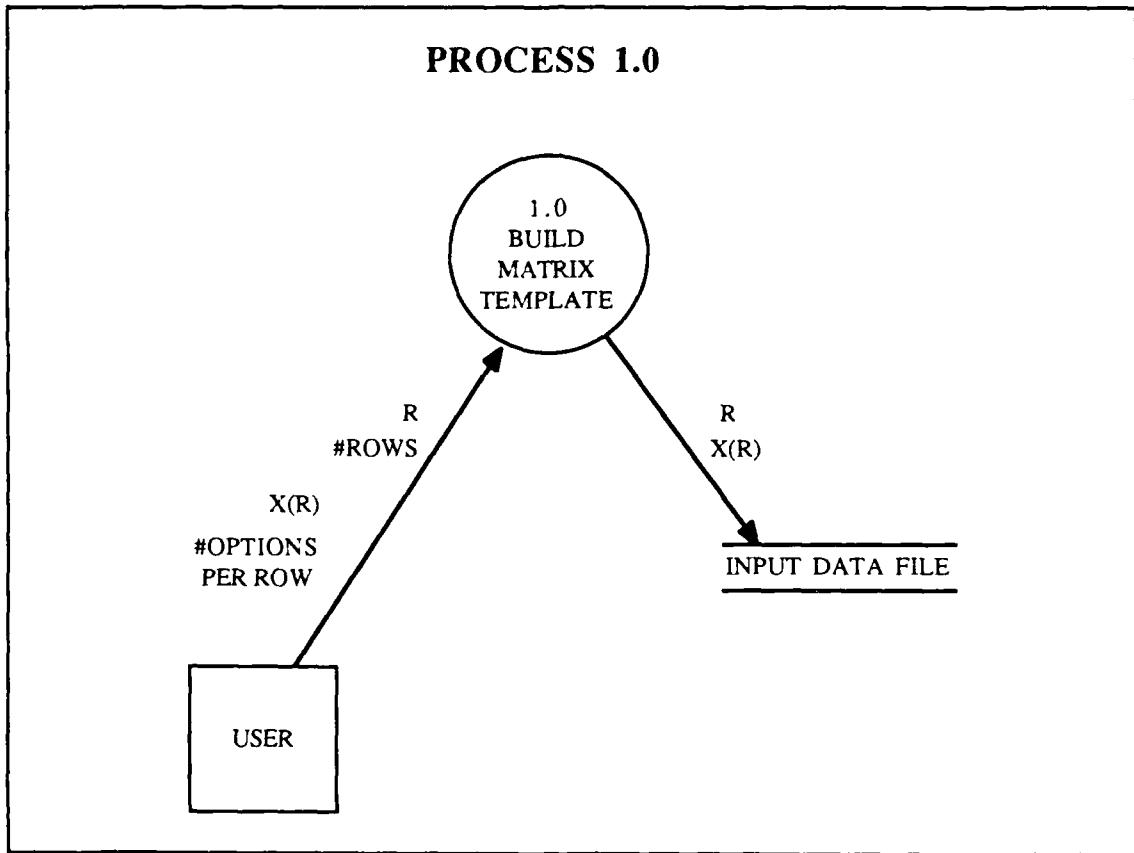
Tisdell, C. A., *Microeconomics: The Theory of Economic Allocation*, New York: Wiley and Sons, 1972, pp. 120-123.

Walsh, V. C., *Introduction to Contemporary Economics*, New York: McGraw-Hill, 1970, pp. 63-66.

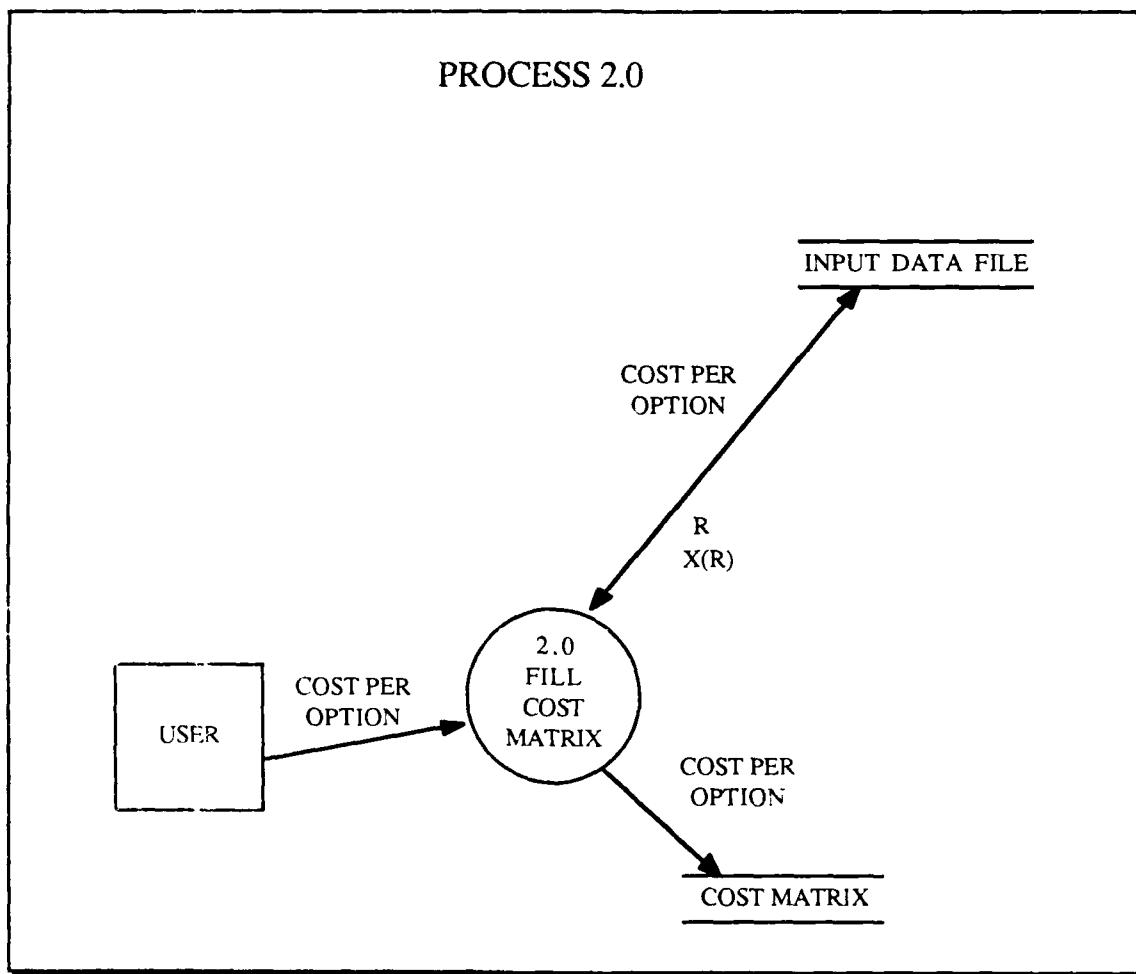
APPENDIX C

DATA FLOW DIAGRAMS

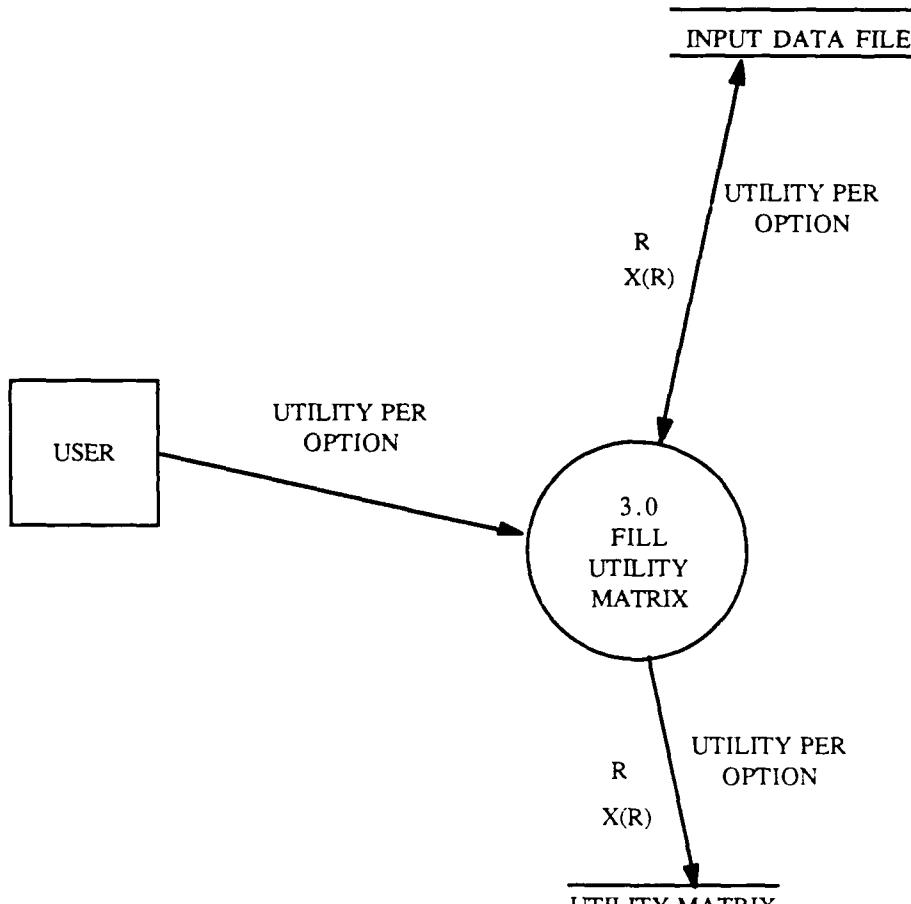
Data flow diagrams (DFD) are the logical models of the processes and data flow for the prototype SCP-DSS. These DFDs do not depend on a particular operating system or hardware configuration but represent an understandable logical model of the overall system.



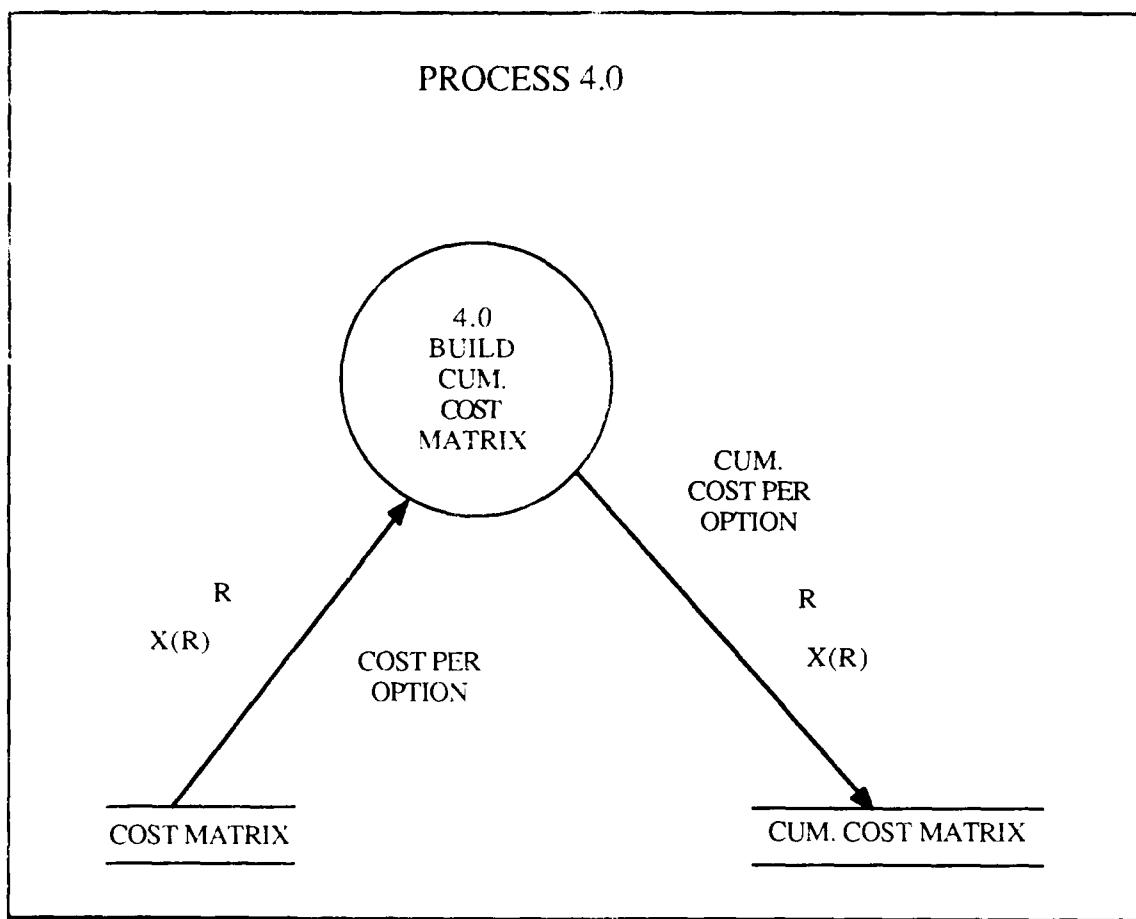
PROCESS 2.0



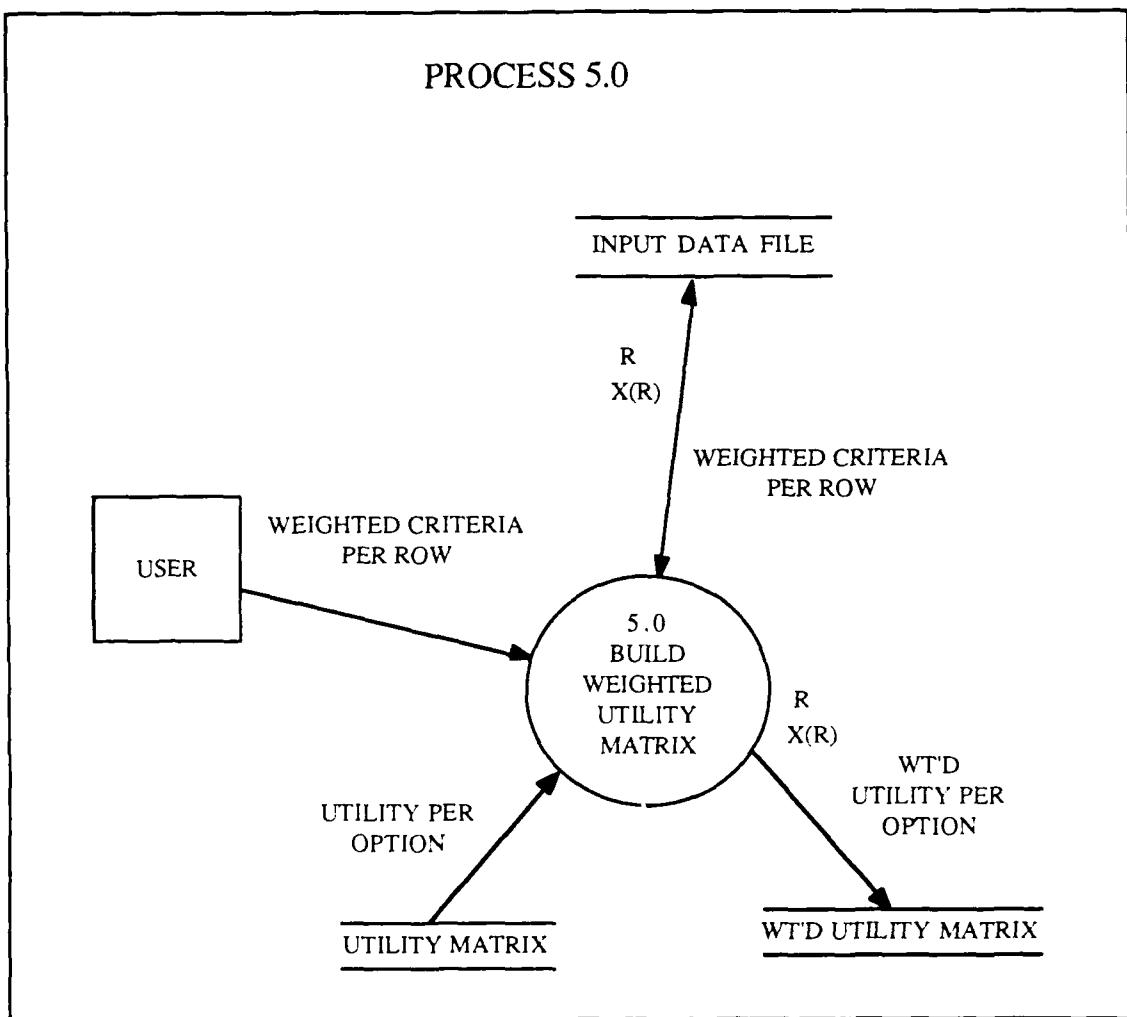
PROCESS 3.0



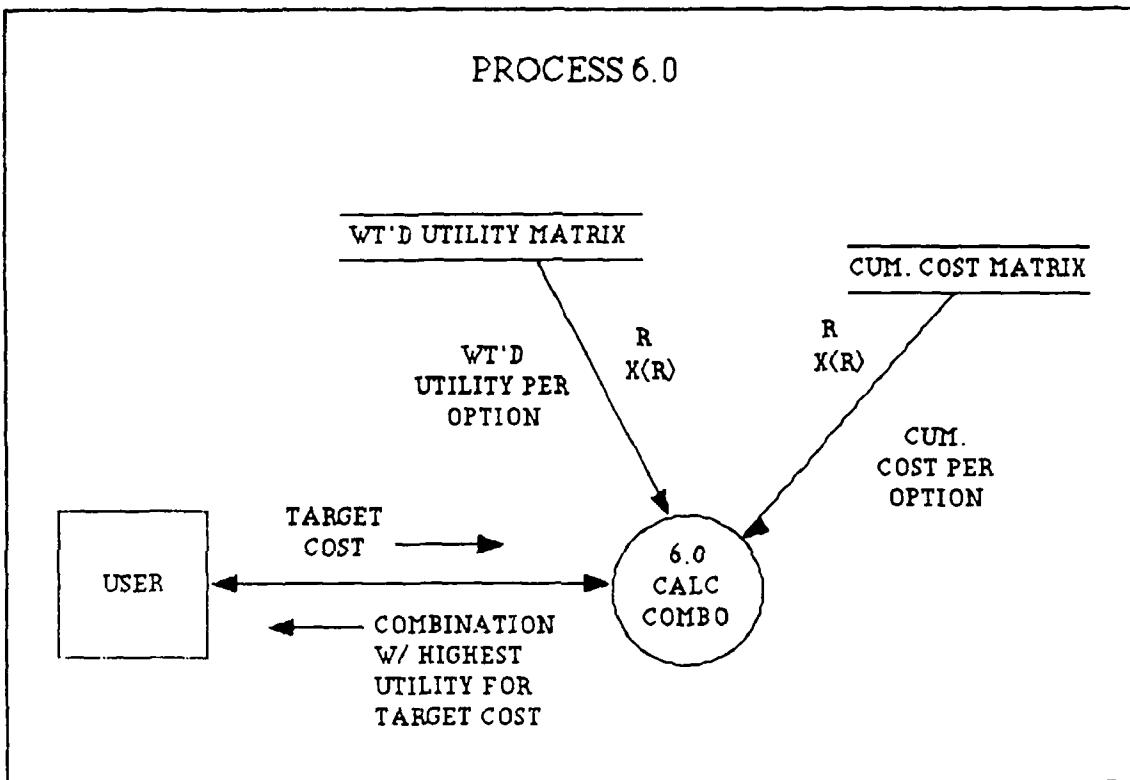
PROCESS 4.0



PROCESS 5.0



PROCESS 6.0



PROCESS 7.0

WT'D UTILITY MATRIX

CUM. COST MATRIX

R
 $X(R)$

CUM.
COST PER
OPTION

WT'D
UTILITY
PER
OPTION

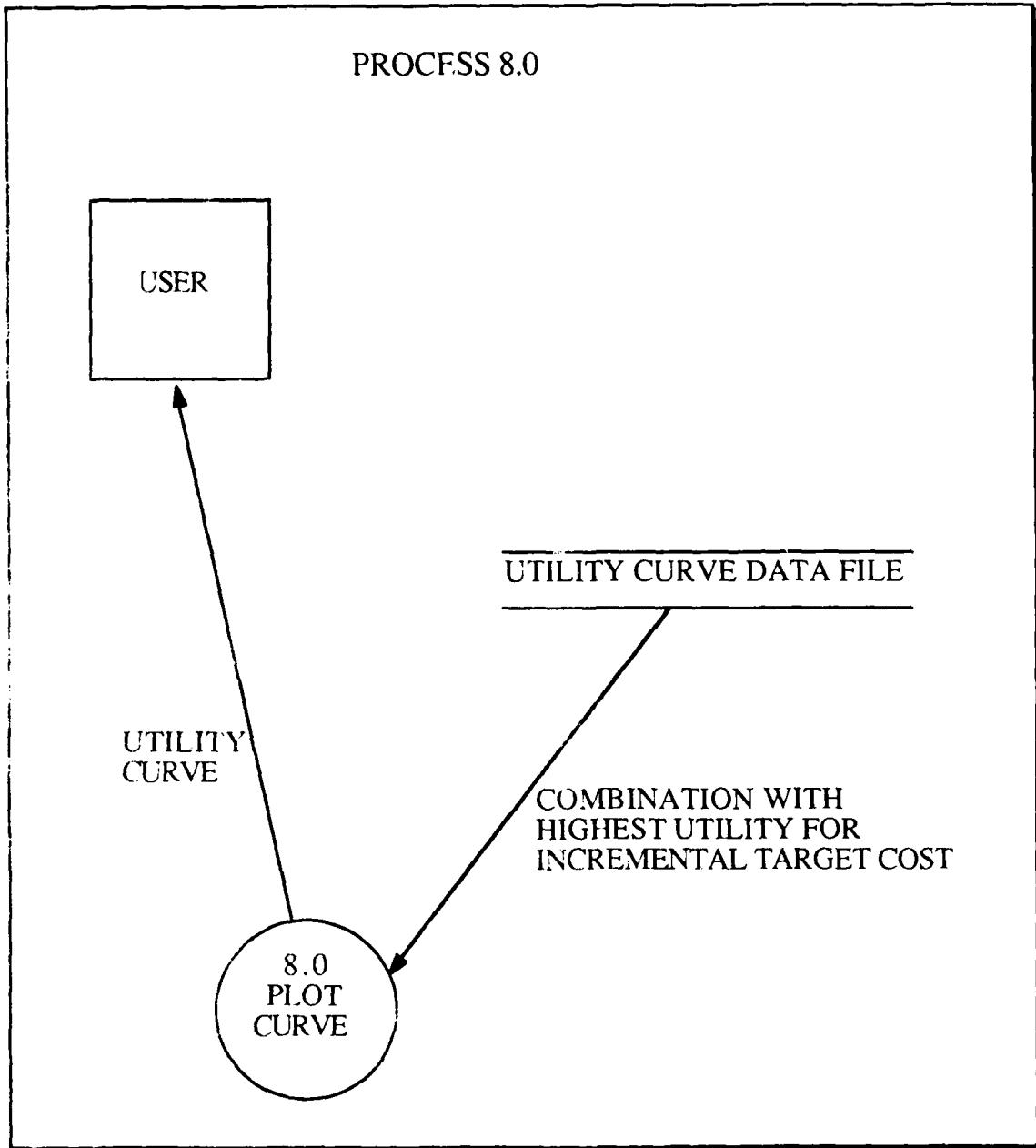
R
 $X(R)$

7.0
CALC
UTILITY
CURVE

COMBINATION WITH
HIGHEST UTILITY FOR
INCREMENTAL TARGET COST

UTILITY CURVE DATA FILE

PROCESS 8.0



APPENDIX D

FIRST PROGRAM LISTING IN PASCAL

program SCP_DSS; {Lt Craig L. Riddle, USN; 1988}

{This source code program listing, in THINK Technologies, Inc.—MacPascal™, is a System Cost Planning Decision support system. It uses applied Multi-attribute Utility theory. The user builds the decision matrix by inputting cost and utility scores for the different combination options. Refer to Thesis Chapter III for decision environment discussion. This program when compiled allows the user to select the functions necessary through menu options. The output can be saved to a file, printed, or both. This code does not allow for the plotting of the utility curve as can be found in Appendix E.}

```
type
  utilplot = array[1..21] of real;
  compname = array[1..15] of string;
  row = array[1..15] of integer;
  matrix = array[1..15] of array[1..15] of integer; {integer matrix format}
  matrixR = array[1..15] of array[1..15] of real; {real value matrix format}

  filedata = record
    r : integer;
    rowopt : row; { matrix rows and options }
    names : compname; { component names file }
    costs : matrix; { input cost data file }
    utilities : matrix; { input utility data file }
    wtutilityfile : row; { relative component weights}
    end;
    savedfile = file of filedata;

var
  ticks : Longint; { keeps track of time }
  R : integer; { variable for number of matrix rows }
  util_mat : matrix; { utility matrix }
  cost_mat : matrix; { cost matrix }
  wt_util_mat : matrixR; { weighted utility matrix }
  cum_cost_mat : matrix; { cumulative cost matrix }
  numopt : row; { number of options per row }
  rowcount : row; { holder for row counts in Docomp_recursion }
  Best : row; { holds best combination of options }
  inputcomp : compname; { component name }
  option : char; { user inputted option }
  rel_comp_wt : row; { relative weight values for components }
  utilpt : utilplot;
```

```

SCP_DSS_data : savedfile;
saverec : filedata;

procedure savefile;
var
  x, y : integer;
begin
  saverec.r := R;
  for x := 1 to R do
    begin
      saverec.rowopt[x] := numopt[x];
      saverec.names[x] := inputcomp[x];
      saverec.wtutilityfile[x] := rel_comp_wt[x];
      for y := 1 to numopt[x] do
        begin
          saverec.costs[x, y] := cost_mat[x, y];
          saverec.utilities[x, y] := util_mat[x, y];
        end;
    end;
  rewrite(SCP_DSS_data);
  write(SCP_DSS_data, saverec);
end;

procedure openfile;
var
  x, y : integer;
begin
  reset(SCP_DSS_data);
  read(SCP_DSS_data, saverec);
  R := saverec.r;
  for x := 1 to R do
    begin
      numopt[x] := saverec.rowopt[x];
      inputcomp[x] := saverec.names[x];
      rel_comp_wt[x] := saverec.wtutilityfile[x];
      for y := 1 to numopt[x] do
        begin
          cost_mat[x, y] := saverec.costs[x, y];
          util_mat[x, y] := saverec.utilities[x, y];
        end;
    end;
end;

procedure get_inp_comp_name;
{ 'Get input component names' procedure allows the user to input the matrix }
{ components names as passed variables for screen, printer, and file}

```

```

{output.}

var
  X, G : integer;

begin
begin
  write('Input the number of components for this matrix - ');
  readln(R);
  writeln;
  for X := 1 to R do

    begin
      write('Input the number of options for row ', X : 3, '- ');
      readln(numopt[X]);
      writeln;
      write('Input component #', X, "'s category name ');
      readln(inputcomp[X]);
      writeln;
    end;
    for X := R + 1 to 15 do
      numopt[X] := 1;
  end;
  writeln;
  sysbeep(3);
  write(' Input 1 to return to the Main Menu ');
  readln(G);
end;

procedure draw_cost_mat;

{ This procedure prints the cost matrix for the user to view}
var
  X, Y, G : integer;

begin
  writeln(' This is the Cost matrix.');
  writeln;
  for X := 1 to R do  {for each row}
    begin
      begin
        for Y := 1 to numopt[X] + 1 do
          write('_____');  {draws the top line for row X}
        writeln;
      end;

      begin
        write(inputcomp[X]); {prints the component name for row X}
      begin

```

```

for Y := 1 to numopt[X] do
  write(' ', cost_mat[X, Y] : 3); {draws cost data for row-X,option-Y}
  writeln(' '); {ends each row}

  for Y := 1 to numopt[X] + 1 do
    write('_____'); {draws bottom line of matrix}
    writeln;
  end;
  end;
  writeln;
  sysbeep(3);
  write(' Input 1 to return to the Main Menu ');
  readln(G);
end;

procedure draw_cum_cost_mat;
{This procedure uses the input cost matrix2 to build and print the cumulative}
{cost matrix for the user. cum_cost_mat is passed to the calculation procedure to}
{gives the cost data for the combination search.}

var
  B, X, Y, G : integer;

begin
  writeln('This is the Cumulative Cost matrix.');
  writeln;
  for X := 1 to R do {for each component row}

    begin
      B := 0;
      for Y := 1 to numopt[X] + 1 do {draws top line of row X for the matrix}
        write('_____');
      writeln;

      begin
        write(inputcomp[X]); {print the component row name for row X}
        begin
          for Y := 1 to numopt[X] do {for each option per row X}

            begin {use cost_mat(cost matrix) to build the cumulative cost matrix}
              B := cost_mat[X, Y] + B;
              write(' ', B : 3);
              cum_cost_mat[X, Y] := B; {cum_cost_mat is calculation procedure to build the
              cumulative cost matrix from the user inputted option cost inputs}
            end;

            writeln(' '); {ends each row}
      end;
    end;
  end;

```

```

for Y := 1 to numopt[X] + 1 do  {draws bottom line of matrix}
  write('_____');
  writeln;
end;
end;
writeln;
sysbeep(3);
write('  Input 1 to return to the Main Menu  ');
readln(G);
end;

procedure get_inp_cost;
{This procedure gets the user cost inputs for each option for all }
{data points. cost_mat is passed to the view cost matrix procedure and}
{as the input data for the cumulative cost matrix procedure}

var
  X, Y, G : integer;

begin
  for X := 1 to R do
    for Y := 1 to numopt[X] do

      begin
        write('Input cost of Component-', X : 3, ' option-', Y : 3, ' of ', numopt[X] : 3, ' ');
        readln(cost_mat[X, Y]); {get user inputted cost data for component}
                                {of row-X, option-Y into cost_mat array}
        writeln;
      end;

      writeln;
      sysbeep(3);
      write('  Input 1 to return to the Main Menu  ');
      readln(G);
    end;
end;

procedure draw_utility_mat;
{This procedure draws the utility matrix for the user to view.}

var
  X, Y, G : integer;

begin
  writeln(' This is the Utility matrix.');
  writeln;
  for X := 1 to R do

```

```

begin
begin {draws top line of each row}
for Y := 1 to numopt[X] + 1 do
  write('_____');
writeln;
end;

begin
write(inputcomp[X]); {prints component name for row X}

begin
for Y := 1 to numopt[X] do
  write(' ', util_mat[X, Y] : 3); {prints option data for each row}
  writeln(' ');
for Y := 1 to numopt[X] + 1 do
  write('_____');
writeln;
end;
end;
writeln;
sysbeep(3);
write(' Input 1 to return to the Main Menu ');
readln(G);
end;

procedure get_inp_utility;
{ This procedure gets the user to input utility scores for each}
{ options per component row. util_mat is the utility input matrix that}
{ is passed to the weighted utility matrix procedure to build the }
{ weighted utility matrix }

var
X, Y, G : integer;

begin
writeln;
for X := 1 to R do {for each row}
begin
  for Y := 1 to numopt[X] do { for each option per row}
    begin
      write('Input utility of Component', X : 3, ' option', Y : 3, ' of ', numopt[X] : 3, ' ');
    end;
    readln(util_mat[X, Y]); {put utility values into coordinates}
    { row-X, option-Y into array util_mat}
    writeln;
end;

```

```

end;

writeln;
sysbeep(3);
write(' Input 1 to return to the Main Menu ');
readln(G);
end;

procedure get_rel_comp_wts;
{ This procedure gets the user to input relative weights for each component. The
inputwtcriteria is passed to the weighted utility matrix procedure }

var
  X, G : integer;

begin
  writeln;
  for X := 1 to R do
    begin
      writeln;
      write('Input relative value of component - ', X : 3, ' - ');
      readln(rel_comp_wt[X]);
      writeln;
      end;
  writeln;
  sysbeep(3);
  write(' Input 1 to return to the Main Menu ');
  readln(G);
end;

procedure draw_wt_utility_mat;
{This procedure uses the input utility matrix and the component relative weights to build
the weighted utility matrix}

var
  sum, X, Y, G : integer;
  outwtcriteria : array[1..15] of real;

begin
  sum := 0;
  for X := 1 to R do
    sum := sum + rel_comp_wt[X]; { adds all input component weights to get sum total}
  for X := 1 to R do

```

```

outwtcriteria[X] := (rel_comp_wt[X] / sum);

{outwtcriteria(output weight criteria) is the percent value the component has as a relative
value to the other components}

begin

writeln("This is the weighted utility matrix.");
writeln;
for X := 1 to R do

begin {drawing weighted utility value matrix}
for Y := 1 to numopt[X] + 2 do {draws the top line of each row to frame the matrix}
  write('_____');
writeln;

begin
  write(outwtcriteria[X] : 3 : 2, inputcomp[X] : 8);
begin
  for Y := 1 to numopt[X] do
  begin
    wt_util_mat[X, Y] := util_mat[X, Y] * outwtcriteria[X];
    write(' | ', wt_util_mat[X, Y] : 3 : 2); {prints the calculated weighted utility matrix
values separated by a vertical bracket}
  end;
  writeln(' |'); {end bracket for each row}

  for Y := 1 to numopt[X] + 2 do {draws the bottom line of the matrix}
    write('_____');
  writeln;

  end;
end;
end; {drawing weighted utility values}
writeln;
sysbeep(3);
write(' Input 1 to return to the Main Menu ');
readln(G);
end;
end;

procedure update_comp_wts;
{This procedure allows the user to change a component weight data point}

var
  A, X, G : integer;

begin

```

```

writeln('In what ROW is the element that you desire to change?-');
readln(X);
writeln('What RELATIVE COMPONENT WEIGHT do you wish to assign to ROW ', X);
readln(A);
rel_comp_wt[X] := A;
writeln;
sysbeep(3);
write(' Input 1 to return to the Main Menu ');
readln(G);
end;

procedure update_cost_inputs;
{This procedure allows the user to change a cost matrix data point}

var
A, X, Y, G : integer;

begin
writeln('In what ROW is the element that you desire to change?-');
readln(X);
writeln('What OPTION NUMBER do you desire to change?-');
readln(Y);
writeln('What COST do you desire to assign to ROW ', X, ' option', Y);
readln(A);
cost_mat[X, Y] := A;
writeln;
sysbeep(3);
write(' Input 1 to return to the Main Menu ');
readln(G);
end;

procedure update_utility_inputs;
{This procedure allows the user to change a utility matrix data point}

var
A, X, Y, G : integer;

begin
writeln('In what ROW is the element that you desire to change?-');
readln(X);
writeln('What OPTION NUMBER do you desire to change?-');
readln(Y);
writeln('What UTILITY VALUE do you desire to assign to ROW ', X, ' option', Y);
readln(A);
util_mat[X, Y] := A;

```

```

writeln;
sysbeep(3);
write(' Input 1 to return to the Main Menu  ');
readln(G);
end;

```

```
procedure find_optimum_combo;
```

{This procedure is the mathematical model component of the SCP-DSS. The user inputs a cost target and the function produces the optimum combination of options with the highest utility score as derived from the cumulative cost matrix and the weighted utility matrix.}

```

var
  X, Y, G, target, range : integer;
  utility, lastutil : real;
  cost, cost1, sum : integer;
  i : integer; {for loop variable}

procedure find_optimum_combo_recursion (Rows : integer);
var
  i, j : integer; {for loop variables}
begin
  for i := 1 to numopt[R - Rows + 1] do
  begin
    rowcount[Rows] := i;
    if Rows > 1 then
      find_optimum_combo_recursion(Rows - 1)
    else
      begin
        cost := 0;
        for j := 1 to R do
          cost := cost + cum_cost_mat[j, rowcount[R - j + 1]];
        if (cost > (target - range)) and (cost < (target + range)) then
          begin
            utility := 0;
            for j := 1 to R do
              utility := utility + wt_util_mat[j, rowcount[R - j + 1]];
            if utility > lastutil then
              begin
                lastutil := utility;
                cost1 := cost;
                for j := 1 to R do
                  Best[j] := rowcount[j];
                end;
              end;
            end;
          end;
      end;
  end; {find_optimum_combo_recursion}

begin
  for X := 1 to R do

```

```

begin
  write(' ');
  for Y := 1 to numopt[X] do
    write(cum_cost_mat[X, Y] : 3, ' ');
  writeln;
end;
sum := 0;
for X := 1 to R do
  sum := cum_cost_mat[X, numopt[X]] + sum;
writeln(' target range is 0 to ', sum);
sysbeep(3);
write(' Input the target COST-');
readln(target);
write(' Input the target COST SEARCH RANGE(plus or minus) i.e 2,3,5,etc.');
readln(range);
writeln(' This may take a few minutes. . . . .');

```

```
find_optimum_combo_recursion(R);
```

```

writeln;
writeln('$ ', cost1);
sysbeep(3);
writeln(' The Optimum Combination of options is:');
write(' ');
for i := 1 to R do
  write(Best[R - i + 1] : 4, ' ');
writeln;
writeln;
writeln(' The highest Utility score is: ', lastutil : 5 : 2);
writeln;
writeln;
sysbeep(3);
write(' Input 1 to return to the Main Menu ');
readln(G);
end;

```

```
procedure find_optimum_combo2;
```

{ This procedure is the mathematical model component of the SCP-DSS used for the utility curve plot. The function produces the optimum combination of options with the highest utility score as derived from the cumulative cost matrix and the weighted utility matrix in twenty incremental steps for the plot of the curve. }

```

type
  utilstep = array[1..21] of integer;
  timearray = array[1..21] of real;

```

```

var
  X, Y, G, M, target : integer;

```

```

utility, lastutil, increment, timesum, rangemax : real;
cost, cost1, sum : integer;
step : utilstep;
clock : timearray;
Str, AVG : real;
i : integer; {for loop variable}

procedure find_optimum_combo2_recursion (Rows : integer);
var
  i, j : integer; {for loop variables}
begin
  for i := 1 to numopt[R - Rows + 1] do
  begin
    rowcount[Rows] := i;
    if Rows > 1 then
      find_optimum_combo2_recursion(Rows - 1)
    else
      begin
        cost := 0;
        for j := 1 to R do
          cost := cost + cum_cost_mat[j, rowcount[R - j + 1]];
        if (cost > (step[X] - rangemax)) and (cost < (step[X] + rangemax)) then
          begin
            utility := 0;
            for j := 1 to R do
              utility := utility + wt_util_mat[j, rowcount[R - j + 1]];
            if utility > lastutil then
              begin
                lastutil := utility;
                cost1 := cost;
                for j := 1 to R do
                  Best[j] := rowcount[j];
                end;
              end;
            end;
          end;
        end;
      end;
    end; {find_optimum_combo2_recursion}

begin
  timesum := 0;
  sum := 0;
  for X := 1 to R do
    sum := cum_cost_mat[X, numopt[X]] + sum;
  increment := (sum / 20);
  rangemax := (sum / 40);
  step[1] := 0;
  for X := 2 to 21 do

```

```

begin
  step[X] := step[X - 1] + round(increment);
end;

writeln('This may take a few minutes. . . . .');
writeln;

for X := 2 to 21 do

begin
  sysbeep(3);
  ticks := tickcount;
  find_optimum_combo2_recursion(R);
  write(' Utility for step ', (X - 1) : 2, ' is: ', lastutil : 5 : 2, ' Cost is $', step[X]:5);
  ticks := tickcount - ticks;
begin
  str := ticks / 60.0; { convert ticks to second }
  writeln(' in ', Str : 5 : 3, ' sec');
  clock[X] := Str;
end;
  write(' The Optimum Combination for step:', (X - 1) : 2);
  write(' - ');
  for i := 1 to R do
    write(Best[R - i + 1] : 4, ' ');
  writeln;
  utilpt[1] := 0;
  utilpt[X] := lastutil;
  lastutil := 0;
end;

for X := 2 to 21 do
  timesum := clock[X] + timesum;
  AVG := timesum / 20;
  writeln(' average time is', AVG : 5 : 3);

writeln;
sysbeep(3);
sysbeep(3);
write(' Input 1 to return to the Main Menu ');
readln(G);

end;

procedure get_choice (var option : CHAR);
{ This procedure prints a menu of options for the user to work or to just quit to terminate
the program}

```

```

procedure Print_Menu (var option : CHAR); {self-explanatory}

begin
begin
writeln;
writeln;
writeln('                                     MENU');
writeln(' INPUT DATA *****');
writeln('      A : MATRIX');
writeln('      B : COST');
writeln('      C : UTILITY');
writeln('      D : COMPONENT WEIGHT');
writeln(' UPDATE DATA *****');
writeln('      E : COST');
writeln('      F : UTILITY');
writeln('      G : COMPONENT WEIGHT DATA');
writeln(' VIEW MATRIX *****');
writeln('      H : COST');
writeln('      I : CUMULATIVE COST');
writeln('      J : UTILITY');
writeln('      K : WEIGHTED UTILITY');
writeln(' FIND COMBINATION *****');
writeln('      L : COMPONENT-OPTION MIX');
writeln('      M : UTILITY CURVE SCORES');
writeln('      O : OPEN SAVED FILE');
writeln('      S : SAVE FILE');
writeln('      Q : QUIT');
writeln;
writeln(' ENTER SELECTION (A, B, C, D, E, F, G, H, I, J, K, L, M, P or
Q) FOLLOWED BY A "RETURN");
write('      ');
sysbeep(3);
read(option);

end; {Print_Menu}
end;

begin {procedure get_choice}
repeat
begin
Print_Menu(option);
case option of
'A' :
get_inp_comp_name; { Gets the # of component rows, number of options
per row, and component names }

'B' :
get_inp_cost; { gets initial option cost inputs}

```

```

'C' :
get_inp_utility;           { gets initial option utility inputs}

'D' :
get_rel_comp_wts;         { gets the relative component weights}

'E' :
update_cost_inputs;       {Update cost data}

'F' :
update_utility_inputs;    {Update utility data}

'G' :
update_comp_wts;          {Update component weight data}

'H' :
draw_cost_mat;             {draws cost matrix}

'I' :
draw_cum_cost_mat;         {draws cumulative cost matrix}

'J' :
draw_utility_mat;          {draws utility matrix}

'K' :
draw_wt_utility_mat;       {draws weighted utility matrix}

'L' :
find_optimum_combo;        {Finds the optimum combination of one option
                           per component that gives the highest utility score based on one user inputted
                           target cost}

'M' :
find_optimum_combo2;        {Finds the optimum combination of one option per
                           component that gives the highest utility scores for 20 data points along the cost axis in
                           equal increments for data points for the utility curve plot}

'O' :
openfile;

'S' :
savefile;

'Q' :
; {quit the program}

otherwise
writeln('Invalid Input -Invalid Input -Invalid Input -Invalid Input');

```

```
end;
end;

until option = 'Q';
writeln('          HAVE A GOOD DAY !');
sysbeep(3);
sysbeep(3);
end;

begin
get_choice(option);
end.
```

APPENDIX E

SECOND PROGRAM LISTING IN PASCAL

program SCP_DSS_TURBO;{Lt Craig L. Riddle, USN; 1988}

{This source code program listing, in Borland International's TURBO™ PASCAL, is a System-Cost Planning Decision support system (SCP_DSS). It uses applied Multi-Attribute Utility Theory (MAUT). The user builds the decision matrix by inputting cost and utility scores for the different combination options. Refer to Thesis Chapter III for decision environment discussion. This program, when compiled, allows the user to select the functions necessary through menu options. The distinguishing characteristics of this code, compared to that in Appendix D, are in the utility curve plotting procedure. There is no provision in this code for saving data to disk nor for printing the plotted graph. }

USES

MEMTYPES, QUICKDRAW, OSINTF, TOOLINTF,PACKINTF;

```
type
  utilplot = array[1..21] of real;
  compname = array[1..15] of string;
  row = array[1..15] of integer;
  matrix = array[1..15] of array[1..15] of integer; {integer matrix format}
  matrixR = array[1..15] of array[1..15] of real; {real value matrix format}

  filedata = record
    rfile    : row;        { matrix rows and options }
    datafile  : compname;  { component names file }
    costfile  : matrix;   { input cost data file }
    utilityfile : matrix; { input utility data file }
    wtutilityfile : row;  { relative component weights }
  end;

  var
    ticks : Longint;          { keeps track of time }
    R : integer;              { variable for number of matrix rows }
    util_mat : matrix;        { utility matrix }
    cost_mat : matrix;        { cost matrix }
    wt_util_mat : matrixR;   { weighted utility matrix }
    cum_cost_mat : matrix;   { cumulative cost matrix }
    numopt : row;             { number of options per row }
    rowcount : row;           { holder for row counts in Docomp_recursion }
    Best : row;               { holds best combination of options }
    inputcomp : compname;    { component name }
    option : char;            { user inputted option }
    rel_comp_wt : row;        { relative weight values for components }
```

```

utilpt : utilplot;
gport : grafport;

procedure get_inp_comp_name ;
  ('Get input component names' procedure allows the user to input the matrix
  components names as passed variables for screen, printer, and file output.)

var
  X,G : integer;

begin
begin
  write('Input the number of components for this matrix - ');
  readln(R);
  writeln;
  for X := 1 to R do

begin
  write('Input the number of options for row ', X : 3, '- ');
  readln(numopt[X]);
  writeln;
  write('Input component #', X, " s category name ");
  readln(inputcomp[X]);
  writeln;
end;
  for X := R + 1 to 15 do
  numopt[X] := 1;
end;
writeln;
sysbeep(3);
write(' Input 1 to return to the Main Menu ');
readln(G);
end;

procedure draw_cost_mat ;
  {This procedure prints the cost matrix for the user to view}
var
  X, Y, G : integer;

begin
  writeln(' This is the Cost matrix.');
  writeln;
  for X := 1 to R do  {for each row}
  begin
  begin
  for Y := 1 to numopt[X] + 1 do
  write('_____'), {draws the top line for row X}

```

```

writeln;
end;

begin
write(inputcomp[X]); {prints the component name for row X}

begin
for Y := 1 to numopt[X] do
  write(' ', cost_mat[X, Y] : 3); {draws cost data for row-X,option-Y}
  writeln(' '); {ends each row}

  for Y := 1 to numopt[X] + 1 do
    write('_____'); {draws bottom line of matrix}
    writeln;
  end;
end;
end;
writeln;
sysbeep(3);
write(' Input 1 to return to the Main Menu ');
readln(G);
end;

```

procedure draw_cum_cost_mat ;

{ This procedure uses the input cost matrix2 to build and print the cumulative cost matrix for the user. cum_cost_mat is passed to the calculation procedure to gives the cost data for the combination search. }

```

var
  B, X, Y, G : integer;

begin
  writeln('This is the Cumulative Cost matrix.');
  writeln;
  for X := 1 to R do {for each component row}

    begin
      B := 0;
      for Y := 1 to numopt[X] + 1 do {draws top line of row X for the matrix}
        write('_____');
        writeln;

      begin
        write(inputcomp[X]); {Print the component row name for row X}
        begin
          for Y := 1 to numopt[X] do {for each option per row X}

            begin {use cost_mat(cost matrix) to build the cumulative cost matrix}

```

```

B := cost_mat[X, Y] + B;
write(' | ', B : 3);
cum_cost_mat[X, Y] := B; {cum_cost_mat is the calculation procedure to build the
cumulative cost matrix given the option cost inputs }
end;

writeln(' |'); {ends each row}

for Y := 1 to numopt[X] + 1 do {draws bottom line of matrix}
  write('_____');
writeln;
end;
end;
writeln;
sysbeep(3);
write(' Input 1 to return to the Main Menu ');
readln(G);
end;

```

procedure get_inp_cost ;

{This procedure gets the user cost inputs for each option for all data points. cost_mat is passed to the view cost matrix procedure and as the input data for the cumulative cost matrix procedure }

```

var
  X, Y, G : integer;

begin
  for X := 1 to R do
    for Y := 1 to numopt[X] do

      begin
        write('Input cost of Component-', X : 3, ' option-', Y : 3, ' of, numopt[X] : 3, ' ');
        readln(cost_mat[X, Y]); {get user inputted cost data for component
          of row-X, option-Y into cost_mat array}
        writeln;
      end;

      writeln;
      sysbeep(3);
      write(' Input 1 to return to the Main Menu ');
      readln(G);
    end;

```

procedure draw_utility_mat ;

{This procedure draws the utility matrix for the user to view.}

```

var
  X, Y, G : integer;

begin
  writeln(' This is the Utility matrix.');
  writeln;
  for X := 1 to R do
    begin
      begin {draws top line of each row}
        for Y := 1 to numopt[X] + 1 do
          write('_____');
        writeln;
      end;

      begin
        write(inputcomp[X]); {prints component name for row X}

        begin
          for Y := 1 to numopt[X] do
            write(' ', util_mat[X, Y] : 3); {prints option data for each row}
          writeln(' ');
        end;
        for Y := 1 to numopt[X] + 1 do
          write('_____');
        writeln;
      end;
      writeln;
      sysbeep(3);
      write(' Input 1 to return to the Main Menu ');
      readln(G);
    end;
  end;

procedure get_inp_utility ;
{ This procedure gets the user to input utility scores for each options per component row.
  util_mat is the utility input matrix that is passed to the weighted utility matrix procedure to
  build the weighted utility matrix }

var
  X, Y, G : integer;

begin
  writeln;
  for X := 1 to R do {for each row}
    begin
      for Y := 1 to numopt[X] do { for each option per row}
        begin

```

```

      write('Input utility of Component-', X : 3, ' option-', Y : 3, ' of, numopt[X] : 3, '
');
      readln(util_mat[X, Y]); {put utility values into coordinates row-X, option-Y
                           into array util_mat}
      writeln;
      end;
      end;

      writeln;
      sysbeep(3);
      write(' Input 1 to return to the Main Menu  ');
      readln(G);
      end;

procedure get_rel_comp_wts ;
{ This procedure gets the user to input relative weights for each component. The
inputwtcriteria is passed to the weighted utility matrix procedure }

var
  X, G : integer;

begin
  writeln;
  for X := 1 to R do
  begin
    write('Input relative value of component - ', X : 3, ' - ');
    readln(rel_comp_wt[X]);
    writeln;
  end;
  writeln;
  sysbeep(3);
  write(' Input 1 to return to the Main Menu  ');
  readln(G);
  end;

procedure draw_wt_utility_mat ;
{ This procedure uses the input utility matrix and the component relative weights to build
the weighted utility matrix }

var
  sum, X, Y, G : integer;
  outwtcriteria : array[1..15] of real;

begin
  sum := 0;
  for X := 1 to R do
    sum := sum + rel_comp_wt[X]; {adds all input component weights to
                                   get sum total}

```

```

for X := 1 to R do
  outwtcriteria[X] := (rel_comp_wt[X] / sum);      {output weight criteria is the percent
value the component has as a relative value to the other components}

begin

writeln('This is the weighted utility matrix.');
writeln;
for X := 1 to R do

begin {drawing weighted utility value matrix}
  for Y := 1 to numopt[X] + 2 do {draws the top line of each row to
frame the matrix}
    write('_____');
  writeln;

begin
  write(outwtcriteria[X] : 3 : 2, inputcomp[X] : 8);
begin
  for Y := 1 to numopt[X] do
  begin
    wt_util_mat[X, Y] := util_mat[X, Y] * outwtcriteria[X];
    write(' | ', wt_util_mat[X, Y] : 3 : 2); {prints the calculated weighted
utility matrix values separated by a vertical bracket}
  end;

  writeln(' |'); {end bracket for each row}

  for Y := 1 to numopt[X] + 2 do {draws the bottom line of the matrix}
    write('_____');
  writeln;

end;
end;
end; {drawing weighted utility values}
writeln;
sysbeep(3);
write(' Input 1 to return to the Main Menu ');
readln(G);
end;
end;

procedure update_comp_wts ;
{This procedure allows the user to change a component weight data point}

var
  A, X, G : integer;

begin
  writeln('In what ROW is the element that you desire to change?-');

```

```

readln(X);
writeln('What RELATIVE COMPONENT WEIGHT do you wish to assign to ROW ',
X);
readln(A);
rel_comp_wt[X] := A;
writeln;
sysbeep(3);
write(' Input 1 to return to the Main Menu ');
readln(G);
end;

procedure update_cost_inputs;
{This procedure allows the user to change a cost matrix data point}

var
A, X, Y, G : integer;

begin
writeln('In what ROW is the element that you desire to change?-');
readln(X);
writeln('What OPTION NUMBER do you desire to change?-');
readln(Y);
writeln('What COST do you desire to assign to ROW ', X, ' option', Y);
readln(A);
cost_mat[X, Y] := A;
writeln;
sysbeep(3);
write(' Input 1 to return to the Main Menu ');
readln(G);
end;

procedure update_utility_inputs ;
{This procedure allows the user to change a utility matrix data point}

var
A, X, Y, G : integer;

begin
writeln('In what ROW is the element that you desire to change?-');
readln(X);
writeln('What OPTION NUMBER do you desire to change?-');
readln(Y);
writeln('What UTILITY VALUE do you desire to assign to ROW ', X, ' option', Y);
readln(A);
util_mat[X, Y] := A;
writeln;
sysbeep(3);
write(' Input 1 to return to the Main Menu ');
readln(G);

```

```

end;

procedure find_optimum_combo;

{This procedure is the mathematical model component of the SCP-DSS. The user inputs a
cost target and the function produces the optimum combination of options with the highest
utility score as derived from the cumulative cost matrix and the weighted utility matrix.}

var
  X, Y, G, target,range : integer;
  utility, lastutil : real;
  cost, cost1, sum : integer;
  i : integer; {for loop variable}

procedure find_optimum_combo_recursion (Rows : integer);
var
  i, j : integer; {for loop variables}
begin
  for i := 1 to numopt[R - Rows + 1] do
  begin
    rowcount[Rows] := i;
    if Rows > 1 then
      find_optimum_combo_recursion(Rows -1)
    else
      begin
        begin
          cost := 0;
          for j := 1 to R do
            cost := cost + cum_cost_mat[j, rowcount[R - j + 1]];
          if (cost > (target - range)) and
            (cost < (target + range)) then
            begin
              utility := 0;
              for j := 1 to R do
                utility := utility + wt_util_mat[j, rowcount[R - j + 1]];
              if utility > lastutil then
                begin
                  lastutil := utility;
                  cost1 := cost;
                  for j := 1 to R do
                    Best[j] := rowcount[j];
                  end;
                end;
            end;
        end;
      end;
  end; {find_optimum_combo_recursion}

begin
  for X := 1 to R do
  begin
    write('  ');
    for Y := 1 to numopt[X] do

```

```

        write(cum_cost_mat[X, Y] : 3, ' ');
        writeln;
        end;
        sum:= 0;
        for X:= 1 to R do
            sum:=cum_cost_mat[X,numopt[X]] + sum;
        writeln(' target range is 0 to ',sum);
        sysbeep(3);
        write(' Input the target COST-');
        readln(target);
        write(' Input the target COST SEARCH RANGE(plus or minus) i.e 2,3,5,etc.');
        readln(range);
        writeln(' This may take a few minutes. . . . .');

        find_optimum_combo_recursion(R);

        writeln;
        writeln('$ ', cost1);
        sysbeep(3);
        writeln(' The Optimum Combination of options is:');
        write(' ');
        for i := 1 to R do
            write(Best[R - i + 1] : 4, ' ');
        writeln;
        writeln;
        writeln(' The highest Utility score is: ', lastutil : 5 : 2);
        writeln;
        writeln;
        sysbeep(3);
        write(' Input 1 to return to the Main Menu ');
        readln(G);
        end;
    
```

procedure find_optimum_combo2 ;

{This procedure is the mathematical model component of the SCP-DSS used for the utility curve plot. The function produces the optimum combination of options with the highest utility score as derived from the cumulative cost matrix and the weighted utility matrix in twenty incremental steps for the plot of the curve.}

```

type
    utilstep = array[1..21] of integer;
    timearray = array[1..21] of real;

var
    X, Y, G, M, target : integer;
    utility, lastutil, increment, timesum, rangemax : real;
    cost, cost1, sum : integer;
    step : utilstep;
    clock : timearray;

```

```

Str,AVG      : real;
i : integer; {for loop variable}

procedure find_optimum_combo2_recursion (Rows : integer);
var
  i, j : integer;      {for loop variables}
begin
  for i := 1 to numopt[R - Rows + 1] do
  begin
    rowcount[Rows] := i;
    if Rows > 1 then
      find_optimum_combo2_recursion(Rows -1)
    else
      begin
        cost := 0;
        for j := 1 to R do
          cost := cost + cum_cost_mat[j, rowcount[R - j + 1]];
        if (cost > (step[X] - rangemax)) and (cost < (step[X] + rangemax)) then
          begin
            utility := 0;
            for j := 1 to R do
              utility := utility + wt_util_mat[j, rowcount[R - j + 1]];
            if utility > lastutil then
              begin
                lastutil := utility;
                cost1 := cost;
                for j:= 1 to R do
                  Best[j] := rowcount[j];
              end;
            end;
          end;
        end;
      end;
  end; {find_optimum_combo2_recursion}

begin
timesum:=0;

sum:= 0;
for X:= 1 to R do
  sum:=cum_cost_mat[X,numopt[X]] + sum;

increment:=(sum/20);
rangemax:=(sum/40);

step[1]:=0;
for X:= 2 to 21 do
begin
  step[X]:=step[X-1] + round(increment);
end;

```

```

writeln('This may take a few minutes. . . . .');
writeln;
lastutil:=0;
for X:= 2 to 21 do

begin
  sysbeep(3);
  ticks:=tickcount;
  find_optimum_combo2_recursion(R);
  write(' Utility for step ',(X-1):2,' is: ', lastutil : 6 : 2,
    ' Cost is $',cost1:5);
  ticks:= tickcount-ticks;
  begin
    Str:=ticks/60.0;           { convert ticks to second  }
    writeln(' in ',Str:5:3,' sec');
    clock[X]:=Str;
  end;
  {write(' The Optimum Combination for step:',(X-1):2);
  write(' - ');
  for i := 1 to R do
    write(Best[R - i + 1] : 4, ' ');
  writeln;}
  utilpt[1]:=0;
  utilpt[X]:=lastutil;
  lastutil:=0;
end;

for X:= 2 to 21 do
  timesum:=clock[X]+timesum;
  AVG:=timesum/20;
  writeln(' average time is', AVG:5:3 );

writeln;
sysbeep(3);
sysbeep(3);
write(' Input 1 to return to the Main Menu ');
readln(G);

end;

procedure utility_curve_plot(var utilpt : utilplot);

type
exch = (yes,no);

var
exchanged : exch;
temp : real;

```

```

X,G,Y : integer;

procedure setupscreen;      {Initializes the display for the curve graphing procedure}

var
    R : rect;
    X : integer;

begin
    openport(@gport);
    R := gport.portrect;
    penpat(white);
    paintrect(R);
    penpat(white);
    framerect(R);
    insetrect(R, 1, 1);
    cliprect(R);

    pensize(1,1);
    penpat(black);

    moveto(40,255);
    for X:=1 to 21 DO
    begin
        line(0,-205);
        moveto(X*20+40,255);
    end;

    moveto(35,250);
    for X:=1 to 11 DO
    begin
        line(405,0);
        moveto(35,250-X*20);
    end;
end; {End setting up the screen for the graphics display}

procedure Dographics;      { The GRAPHICS ROUTINE that plots the utility
curve using the utility point data generated in the procedure
find_optimum_combo2_recursion}

var
    X,Y,Y0,DX : integer;
    NS : STR255;

begin
    pensize(1,1);

```

```

penpat(black);

  with screenbits.bounds do
begin
  moveto(40,250);
  Y0:= 0;
  DX:=20;
  for X:=1 to 20 do
  if utilpt[X]<>0 then
    begin
      Y:= ROUND(-(utilpt[X])/100*200);
      line(DX,Y-Y0);

      Y0:=Y;
      DX:=20;
    end
  else
    DX:=DX+20;
end;

begin
Textfont(4);
Textsize(9);
moveto (50,20);
drawstring('CLICK MOUSE TO RETURN TO MENU');
moveto (175,40);
drawstring('PLOTTED UTILITY CURVE');

for X:=0 to 10 DO
begin
  numtostring(X*10,NS);
  Y:= stringwidth(NS);
  moveto(X*40+40-ROUND(Y/2),270);
  drawstring(NS);
end;

for X:=0 to 10 DO
begin
  numtostring(X*10,NS);
  Y:= stringwidth(NS);
  moveto(33-Y,255-X*20);
  drawstring(NS);
end;

repeat
  until button
end;
end;

```

```

begin
  initgraf(@THEPORT);
  initfonts;
  initcursor;
  hidcursor;
  flushevents(EVERYEVENT, 0);
  setupscren;
  dographics;
  sysbeep(3);
end;

procedure get_choice (var option : CHAR);

{ This procedure prints a menu of options for the user to work or to just quit to terminate
the program}

procedure print_menu (var option : CHAR); {self-explanatory}
begin
begin
writeln;
writeln;
writeln('                                     MENU');
writeln(' INPUT DATA *****');
writeln('     A : MATRIX');
writeln('     B : COST');
writeln('     C : UTILITY');
writeln('     D : COMPONENT WEIGHT');
writeln(' UPDATE DATA *****');
writeln('     E : COST');
writeln('     F : UTILITY');
writeln('     G : COMPONENT WEIGHT DATA');
writeln(' VIEW MATRIX *****');
writeln('     H : COST');
writeln('     I : CUMULATIVE COST');
writeln('     J : UTILITY');
writeln('     K : WEIGHTED UTILITY');
writeln(' FIND COMBINATION *****');
writeln('     L : COMPONENT-OPTION MIX');
writeln('     M : UTILITY CURVE SCORES');
writeln('     P : PLOT UTILITY CURVE');
writeln('     Q : QUIT');
writeln;
writeln(' ENTER SELECTION (A, B, C, D, E, F, G, H, I, J, K,
L, M, P or Q) FOLLOWED BY A "RETURN");
write('      ');
sysbeep(3);
read(option);

end; {print_menu}
end;

```

```

begin {procedure get_choice}
repeat
begin
print_menu(option);
CASE option OF
'A' : get_inp_comp_name;           { Gets the # of component rows,
number of options per row, and component names }

'B' : get_inp_cost;                { gets initial option cost inputs}

'C' : get_inp_utility;             { gets initial option utility inputs}

'D' : get_rel_comp_wts;           { gets the relative component weights}

'E' : update_cost_inputs;         {Update cost data}

'F' : update_utility_inputs;       {Update utility data}

'G' : update_comp_wts;            {Update component weight data}

'H' : draw_cost_mat;              {draws cost matrix}

'I' : draw_cum_cost_mat;          {draws cumulative cost matrix}

'J' : draw_utility_mat;           {draws utility matrix}

'K' : draw_wt_utility_mat;         {draws weighted utility matrix}

'L' : find_optimum_combo;         {Finds the optimum combination of one
option per component that gives the highest utility score based on one user
inputted target cost}

'M' : find_optimum_combo2;         {Finds the optimum combination of one option
per component that gives the highest utility scores for 20 data points along the
cost axis in equal increments for data points for the utility curve plot}

'P' : utility_curve_plot(utilpt);  {Plots the utility curve using the saved utility
scores stored in the utilpt array}

'Q' : ; {quit the program}
otherwise
writeln('Invalid Input -Invalid Input -Invalid Input -Invalid Input');
sysbeep(3);
sysbeep(3);
sysbeep(3);
sysbeep(3);
end;
end;

```

```
until option = 'Q';
  sysbeep(3);
  sysbeep(3);
end;

begin
  get_choice(option);
end.
```

LIST OF REFERENCES

1. Dickson, Gary H., and Wetherbe, James C., *The Management of Information Systems*, McGraw-Hill Book Company, New York, 1985, p. 220.
2. Thierauf, Robert J., *Decision Support Systems, for Effective Planning and Control*, McGraw-Hill Book Company, New York, 1982, p. 57.
3. Sprague, Ralph H., and Carlson, Eric D., *Building Effective Decision Support Systems*, Prentice-Hall, Inc., Englewood Cliffs, NJ, 1982, 329 pp.
4. Turban, E., and Meredith, Jack R., *Fundamentals of Management Science*, BPI, Plano, TX, 1985, pp. 88.
5. Sprague and Carlson, eds., *Decision Support Systems: Putting Theory into Practice*, Prentice-Hall, Inc., Englewood Cliffs, NJ, 1986, ch. 1.
6. Thierauf, Robert J., *Decision Support Systems for Effective Planning and Control*, McGraw-Hill Book Company, New York, 1982, p. 59.
7. Sprague, R. H., Jr., "A Framework for the Development of Decision Support Systems," *MIS Quarterly*, December 1980.
8. Gorry, G., and Scott-Morton, Michael S., "A Framework for Management Information Systems," *Sloan Management Review*, 1971.
9. Raitt, Robert A., "Must We Revolutionize Our Methodology?" *Interfaces*, Vol. 4, No. 2, February 1974.
10. March, J. G., and Simon, H. A., *Organizations*, New York: Wiley and Sons, Inc., 1958.
11. Thierauf, Robert J., *Decision Support Systems for Effective Planning and Control*, McGraw-Hill Book Company, New York, 1982, pp. 86-87.
12. Simon, H. A., *Models of Man: Social and Rational*, New York: Wiley and Sons, Inc., 1957.
13. Von Neumann, J., and Morgenstern, O., *Theory of Games and Economic Behavior*, Princeton University Press: Princeton, NJ, 1944.
14. Lucas, H. C., Jr., *The Analysis, Design, and Implementation of Information Systems*, 2nd ed., McGraw-Hill Book Company, New York, 1981.
15. Mason, Richard O., "Basic Concepts for Designing Management Information Systems," 1969, in *Measurement for Management Decision*, Addison-Wesley Publishing Company, Inc., MA, 1981.

16. Jenkins, A. M., "A Framework For MIS Research," *Proceedings of the 9th Annual Conference: American Institute for Decision Sciences*, Chicago, October 1977, p. 573.
17. Dickson, Gary H., and Wetherbe, James C., *The Management of Information Systems*, McGraw-Hill Book Company, New York, 1985, pp. 347-348.
18. Luqi, M. K., "A Computer Aided Prototyping System," submitted to *First International Workshop on Computer-Aided Software Engineering*, May 1987.
19. Koffman, E. B., *Problem Solving and Structured Programming in PASCAL*, 2nd ed., Addison-Wesley Publishing Company, Inc., MA, 1985, p. 11.
20. Stadler, W., "Preference Optimality (On Optimality Concept in Multicriteria Problems)," in W. Oettli and K. Ritter (eds.), *Optimization and Operations Research*, pp. 129-306, Springer-Verlag, New York, 1976.
21. Hwang, C., and Yoon, K., "Multiple Attribute Decision Making: Methods and Applications," in M. Beckmann and H. P. Kunzi (eds.), *Lecture Notes in Economics and Mathematical Systems*, p. 208, Springer-Verlag, New York, 1981.
22. Keeney, R. L., "Quasi-Separable Utility Functions," *Naval Research Logistics Quarterly*, vol. 15, no. 4, pp. 551-556, 1968.
23. Keeney, R. L., "Utility Independence and Preferences for Multiattributed Consequences," *Operations Research*, vol. 19, no. 4, pp. 875-893, 1971.
24. Keeney, R. L., "Utility Functions for Multiattributed Consequences," *Management Science*, vol. 18, no. 5, part 1, pp. 276-287, 1972.
25. Keeney, R. L., "Concepts of Independence in Multiattribute Utility Theory," in J. Cochran and M. Zeleny (eds.), *Multiple Criteria Decision Making*, University of South Carolina Press, Columbia, South Carolina, 1973.
26. Keeney, R. L., "Multiplicative Utility Functions," *Operations Research*, vol. 22, no. 1, pp. 22-34 1974.
27. Farquhar, P. H., "A Survey of Multiattribute Utility Theory and Applications," in M. Starr and M. Zeleny (eds.), *Multiple Criteria Decision Making*, North Holland, New York, 1977.
28. Fischer, G. W., "Experimental Applications of Multi-Attribute Utility Models," in D. Wendt and C. Vlek (eds.), *Utility, Probability, and Human Decision Making*, D. Reidel Pub. Co., Boston, 1975.
29. Fischer, G. W., "Utility Models for Multiple Objective Decisions: Do They Accurately Represent Human Preferences?" *Decision Sciences*, vol. 10, no. 3, pp. 451-479, 1979.
30. Fishburn, P. C., "Lexicographic Orders, Utilities, and Decision Rules: A Survey," *Management Science*, vol. 20, no. 11, pp. 1442-1471, 1974.

31. Fishburn, P. C., "A Survey of Multiattribute/ Multicriterion Evaluation Theories," in S. Zionts (ed.), *Multiple Criteria Decision Making: Kyoto 1975*, Springer-Verlag, New York, 1976.
32. Keeney, R. L., and Raiffa, H., *Decision with Multiple Objectives: Preferences and Value Tradeoffs*, New York: Wiley and Sons, Inc., 1976.

INITIAL DISTRIBUTION LIST

	<u>No. Copies</u>
1. Defense Technical Information Center Cameron Station Alexandria, VA 22304-6145	2
2. Library, Code 0142 Naval Postgraduate School Monterey, CA 93943-5002	2
3. Department Chairman, Code 54 Department of Administrative Sciences Naval Postgraduate School Monterey, CA 93943-5100	1
4. Dr Gerald R. Paurer, Code 55Pa Department of Operations Research Naval Postgraduate School Monterey, CA 93943-5100	4
4. Assistant Professor Barry Frew, Code 54Fw Department of Administrative Sciences Naval Postgraduate School Monterey, CA 93943-5100	1
5. Computer Technology Programs, Code 37 Naval Postgraduate School Monterey, CA 93943-5100	1
6. LT C. L. Riddle, USN 456 "B" Ave. Coronado, CA 92118	2